ENTERPRISE ARCHITECT

Série de Guides d'Utilisateur

Statemachines Exécutables

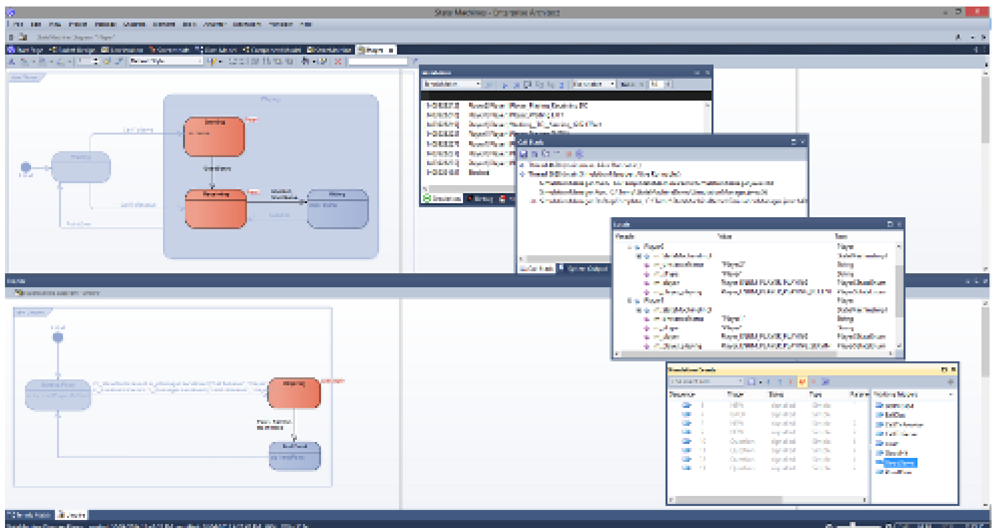
Author: Sparx Systems
Date: 7/11/2024

Version: 17.0

Table des Matières

Statemachines Exécutables 3
Modélisation Statemachines Exécutables 5
Artefact Statemachine Exécutable 10
Génération de code pour Statemachines Exécutables 12
Débogage de l'exécution des Statemachines Exécutables 18
Exécution et Simulation de Statemachines Exécutables 20
Exemple : Statemachine Exécutable 21
Exemple : Commandes de Simulation 25
Exemple : Simulation en HTML avec JavaScript 33
Lecteur CD 34
Parser d'expressions régulières 38
Exemple : Entrer d'un State 40
Exemple : Fourche et Joindre 49
Exemple : Motif d'événement différé 53
Exemple : points d'entrée et de sortie (références de points de connexion) 59
Exemple : Pseudo-state Historique 64
Macro d'événement : EVENT_PARAMETER 72

Statemachines Exécutables



Statemachines Exécutables permettent de générer, d'exécuter et de simuler rapidement des modèles d'état complexes. Contrairement à la simulation dynamique de diagrammes State à l'aide du moteur Simulation d' Enterprise Architect , Statemachines Exécutables fournissent une implémentation complète spécifique au langage qui peut former le « moteur » comportemental de 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 exigences particulières, chaque langage est fourni avec un ensemble de gabarits de code. Vous pouvez personnaliser Gabarits pour adapter le code généré de la manière qui vous convient.

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

Présentation de la construction et de l'exécution Statemachines

Créer et utiliser Statemachines Exécutables est assez simple, mais nécessite un peu de planification et quelques connaissances sur la manière de lier les différents composants pour créer un modèle d'exécution efficace. Heureusement, vous n'avez pas besoin de passer des heures à obtenir le bon modèle et à corriger les erreurs de compilation avant de pouvoir commencer à visualiser votre conception.

Après avoir esquissé les grandes lignes 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 nécessaire pour commencer à exécuter et à simuler Statemachines .

Facilité	Description
Construire des modèles de classes 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 qui vous intéresse 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 classes et vos modèles State , il est temps de concevoir l'artefact Statemachine Exécutable . Celui-ci décrira les classes et les objets impliqués, ainsi que leurs propriétés et relations initiales. C'est le script de liaison qui lie plusieurs objets entre eux et détermine comment ceux-ci communiqueront au moment de l'exécution. Note qu'il est possible d'avoir deux ou plusieurs objets 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 offrent un outil efficace, vous permettant de régénérer l'exécutable à tout moment, et sans perte du code personnalisé que vous auriez pu créer. Il s'agit là d'un avantage majeur sur la durée de vie d'un projet. Il convient également de noter que l'ensemble de la base de code générée est indépendante et portable. Le code n'est en aucun cas couplé à une 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 plates-formes. Vous pouvez également contrôler la vitesse à laquelle les machines fonctionnent pour mieux comprendre la chronologie des événements.
Déboguer Statemachines	Lorsque les états doivent changer mais ne le font pas, lorsqu'une transition ne doit pas être activée mais l'est, lorsque le comportement est - en bref - indésirable et n'est pas immédiatement apparent à partir du modèle, nous pouvons nous tourner vers le débogage. L' Analyseur d'Exécution Visuelle d' Enterprise Architect est fourni 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 peut 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.

Modélisation Statemachines Exécutables

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

- Le langage de code (JavaScript , C# , Java, C++ y compris C)
- Les classes et Statemachines impliquées dans le scénario
- Les spécifications d'instance, y compris l'état d'exécution ; note que cela peut 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 Statemachine modélisées. Par exemple, dans un scénario de match de tennis simple, vous pouvez définir une classe pour chaque joueur, match, Hit et arbitre. Chacun aura sa ou ses propres Statemachine et sera représenté au moment de l'exécution par des instances object pour chaque entité impliquée. Consultez 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 légal 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 autre 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 démarrage elles fonctionnent. Du point de vue de la configuration, l'Artifact est utilisé pour établir le lien vers un script d'analyse qui déterminera le répertoire de sortie, le langage de code, le script de compilation et autres. Un clic droit sur l'Artifact 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

<table>
<tr><td>Statemachines</td><td>
Statemachine simple : la Statemachine a une région
Statemachine orthogonale : la Statemachine contient plusieurs régions
Sémantique d'activation de la région de niveau supérieur (appartenant à Statemachine) :
Activation par défaut : lorsque la Statemachine commence à s’exécuter.
Point d'entrée 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 à un sommet dans cette région
Note 2 : cette Statemachine peut être référencée par un State de sous-machine - les références de point de connexion doivent être définies dans l' State de sous-machine comme sources/cibles de transitions ; la référence de point de connexion représente une utilisation d'un point d'entrée/sortie défini dans la Statemachine et référencé par l' State de sous-machine
Statemachines multiples : L'ordre d'affichage 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 ; celle du haut sera définie comme Statemachine principale
Non pris en charge
Statemachine de protocole
Redéfinition Statemachine
</td></tr>
<tr><td>States</td><td>
State simple : n'a pas de sommets ou de transitions internes
State composite : contient exactement une région
State orthogonal : contient plusieurs régions
State de sous-machine : fait référence à une Statemachine entière
</td></tr>
<tr><td>Entrée State composite</td><td>
Entrée par défaut
Entrée explicite
Entrée d'histoire superficielle
Entrée d'histoire profonde
Point d'entrée Entrée
</td></tr>
<tr><td>Sous-États</td><td>
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).
</td></tr>
<tr><td>Support aux transitions</td><td>
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
</td></tr>
</table>

	• Priorités de tir et algorithme de sélection Pour plus de détails, reportez-vous à la *Spécification UML UML* .
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 modification, vous devez définir un mécanisme externe pour générer des signaux en fonction de 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 : public Signal{
	publique:
	Signal2(){};
	Signal2(std::vector<String>& lstArguments);
	int maVal;
	};
	Note : des détails supplémentaires 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. Il est la source d'au plus une transition ; il ne peut y avoir qu'un seul sommet initial dans une région.
Régions	**Activation par défaut et activation explicite :** Les transitions se terminent sur l' State contenant : • Si un pseudo-état initial est défini dans la région : **activation par défaut** • Si aucun pseudo-état initial n'est défini, la région restera inactive et l' State qui la contient sera 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 du même pseudo-état de fourche) Par exemple, s'il existe trois régions définies pour un State orthogonal et que *les régions A* et *B* ont un pseudo-état initial, alors *la région C* est explicitement activée. L'activation par défaut s'applique aux *régions A* et *B* ; l' State contenant aura trois régions actives.
Choix	Les contraintes de garde sur toutes les transitions sortantes sont évaluées de manière dynamique lorsque la traversée de transition composée atteint ce pseudo-état.
Jonction	Branche conditionnelle statique : les contraintes de garde sont évaluées avant

	l'exécution de toute transition composée.
Fourcher / Rejoindre	Non threadé, chaque région active se déplace d'une étape en alternance, en fonction d'un mécanisme de pool d'événements d'achèvement.
Nœuds de point d'entrée/point de sortie	Non threadé pour State orthogonal ou Statemachine orthogonale ; chaque région active se déplace d'une étape en alternance, en fonction d'un mécanisme de pool d'événements d'achèvement.
Nœuds d'histoire	• DeepHistory : représente la configuration State active la plus récente de son State propriétaire • ShallowHistory : représente le sous-état actif le plus récent de l' State qui le contient, mais pas les sous-états de ce sous-état
Événements différés	Dessinez une transition automatique et modifiez le *type* de transition en interne. Type « defer(); » 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 State de sous-machine) d'un point d'entrée/sortie défini dans la Statemachine référencée par l' State de sous-machine. Les références de point de connexion d'un State de sous-machine peuvent être utilisées comme sources et cibles de transitions. Elles représentent des entrées ou des sorties de la Statemachine référencée par l' State de sous-machine.
Comportements State	Les comportements State « entrée », « doActivity » et « sortie » 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 simultané avec d'autres comportements d'entrée ; le comportement « doActivity » est implémenté comme comportement « exécuter dans la 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 actuels, vous devez ajouter un connecteur <<import>> de la classe de contexte actuelle à la classe de contexte du conteneur. Par exemple :

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

Par conséquent, nous ajoutons un connecteur <<import>> de la classe 1 à la classe 2 afin que la génération de code Statemachine Exécutable génère correctement le code pour State sous-machine S1. (Sur la classe 1, cliquez sur la flèche Quick Linker et faites-la 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, peut être réutilisé pour créer à la fois une résistance à feuille et une résistance à fil enroulé. Cela est susceptible d'être 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' peut être tout ce qui est requis du point de vue de modélisation . Les mêmes Statemachines peuvent ensuite être réutilisées pour tester les changements de comportement qui pourraient résulter de la variation de l'état d'exécution. Voici la procédure :

Étape	Action
Créer ou ouvrir diagramme de composants	Ouvrez un diagramme de composant sur lequel travailler. Il peut s'agir du diagramme qui contient votre artefact d'origine.
Sélectionnez l' 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 d'origine sur votre diagramme . Deux questions vous seront posées. La réponse à la première question est **Object** et à la seconde, **Tout.** Renommez l'Artefact pour le différencier de l'original, puis modifiez ses valeurs de propriété.

Artefact Statemachine Exécutable

Un artefact Statemachine Exécutable est essentiel pour générer Statemachines qui peuvent interagir les unes avec les autres. Il spécifie les objets qui seront impliqués dans une simulation, leur état et la manière dont ils se connectent. L'un des grands avantages de l'utilisation d'artefacts Statemachine Exécutable est que chacune des différentes parties d'un artefact peut représenter une instance d'une Statemachine . Vous pouvez donc configurer des simulations à l'aide de plusieurs instances de chaque Statemachine et observer comment elles interagissent. Un exemple est fourni dans la rubrique d'aide *Exemple d' Statemachine Exécutable* .

Création des Propriétés d'une Statemachine Exécutable

Chaque scénario Statemachine Exécutable implique une ou plusieurs Statemachines . Les Statemachines incluses sont spécifiées par des éléments de propriété UML ; chaque propriété aura un classificateur UML (classe) qui détermine la ou Statemachine incluses pour ce type. Plusieurs types inclus en tant que plusieurs Propriétés peuvent finir par inclure de nombreuses Statemachines , qui sont toutes créées dans le code et initialisées à l'exécution.

Action	Description
Déposez une classe de la fenêtre Navigateur sur l'artefact << Statemachine Exécutable >>	La façon la plus simple de définir des propriétés sur un Statemachine Exécutable est de déposer la classe sur l' 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 la manière dont l' 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 cochant la case « Maintenir la touche Ctrl enfoncée 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 de la même Statemachine , de différentes Statemachines pour la même instance ou Statemachines complètement différentes de différents types de base. Pour créer plusieurs propriétés qui utiliseront la même Statemachine , déposez la même classe sur l'artefact plusieurs fois. Pour utiliser différents types, déposez différentes classes à partir de la fenêtre Navigateur selon vos besoins.

Définition de l' State initial des Propriétés

Les Statemachines exécuter par un Statemachine Exécutable exécuter toutes dans le contexte de leur propre instance de classe. 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 d'autres propriétés similaires d'un joueur si ces propriétés sont pertinentes pour le scénario en cours exécuter . En procédant ainsi, il est possible de définir des conditions initiales détaillées qui influenceront le déroulement du scénario.

Action	Description
Dialogue Définir les valeurs des propriétés	La dialogue d'attribution de 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 des propriétés » 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 saisissez 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 référencer d'autres propriétés en fonction du modèle de classe dont elles sont des instances.

Action	Description
Créer un connecteur	Connectez plusieurs propriétés à l'aide de la relation Connecteur de la boîte à outils Composite. Connector Vous pouvez également utiliser le Quick Linker pour créer une relation entre deux Propriétés et sélectionner « Connecteur » comme type de relation.
Carte de 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 ce faire, 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 pendant l'exécution.

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 utilisé, 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, tout Statemachine Exécutable peut être généré, construit et exécuté en un clic.

Langue prise en charge

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

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

À partir de la version 14.1 Enterprise Architect , la génération de code est prise en charge sans dépendance vis-à-vis de l'environnement de simulation (compilateurs). Par exemple, si vous n'avez pas installé Visual Studio, 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 d'accès au cadre
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)
Microsoft .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)

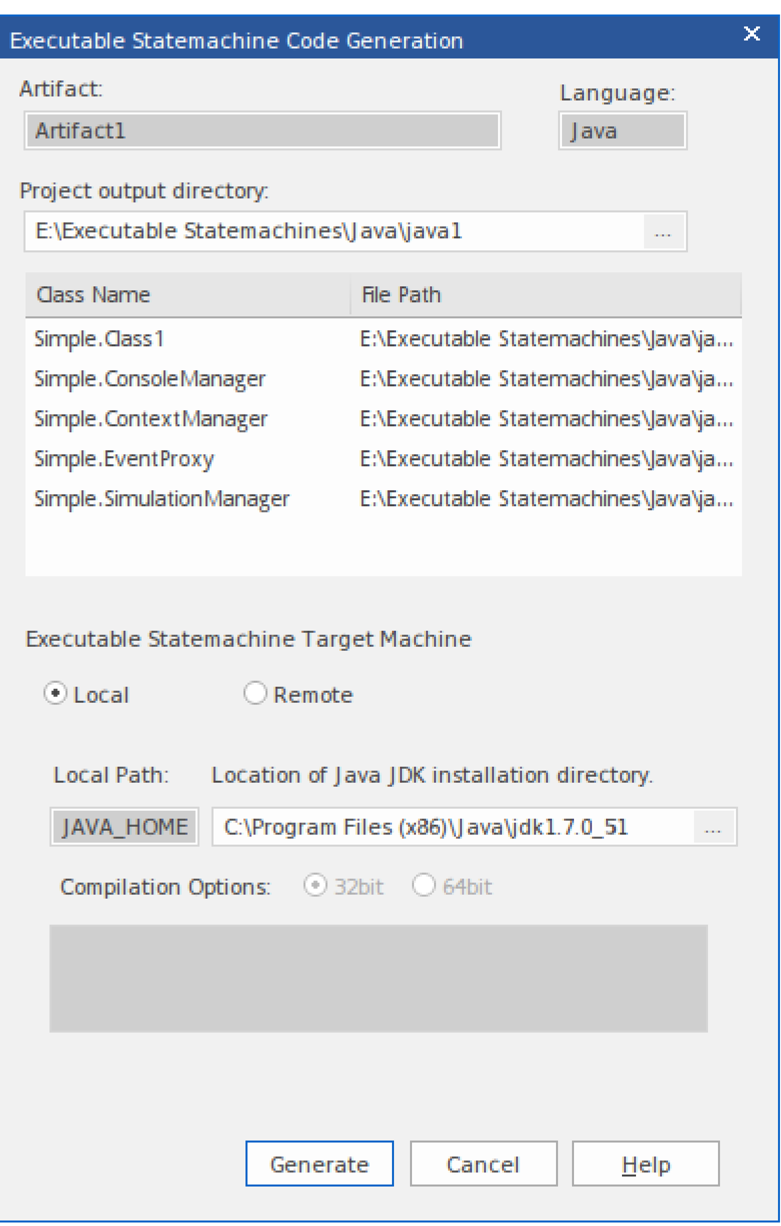
Accéder

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

Génération de code

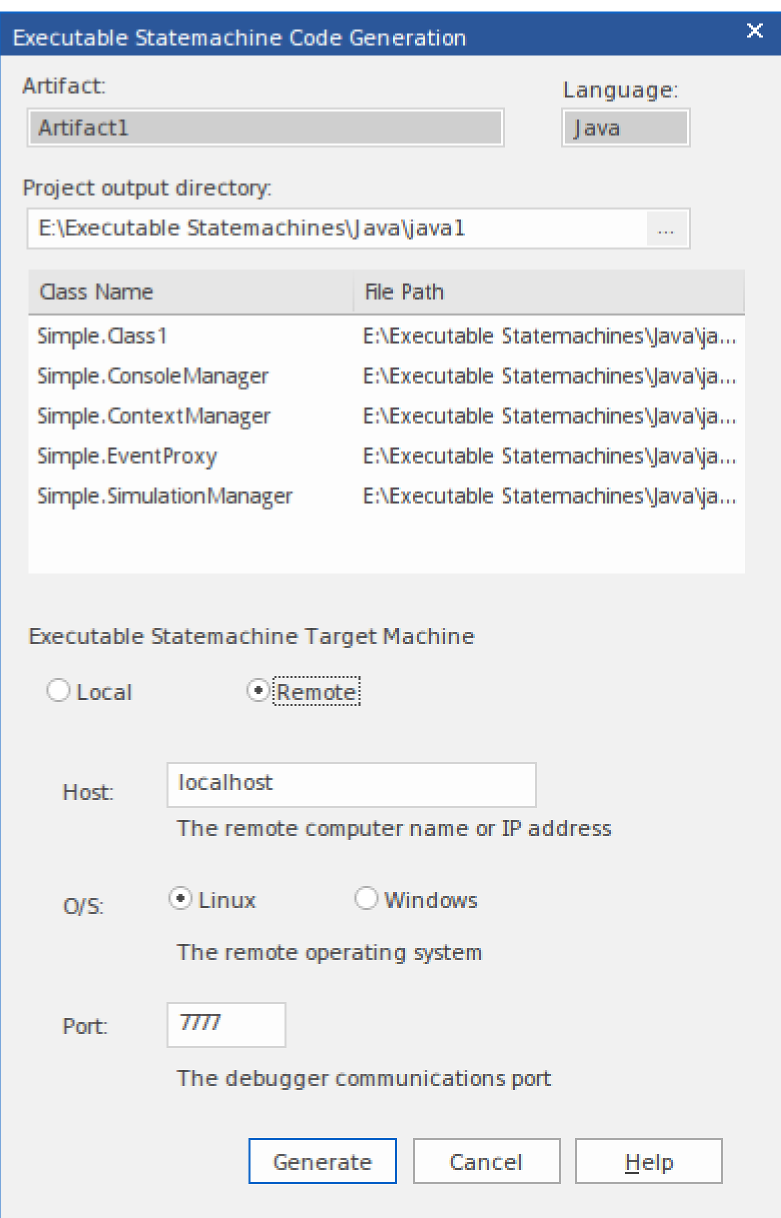
Les options du ruban « Simuler > States Exécutables > Statemachine » fournissent des commandes pour générer du code pour la Statemachine . Sélectionnez d'abord l'artefact Statemachine Exécutable , puis utilisez l'option du ruban pour générer le code. La dialogue « Génération de code de Statemachine exécutable » affichée dépend du langage de code.

Génération de code (Java sur 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 d'accès aux fichiers sources s'affichent ensuite.
Machine Statemachine exécutable Machine cible	Sélectionnez l'option « Local ».
JDK Java	Entrez le répertoire d'installation du JDK Java à utiliser.

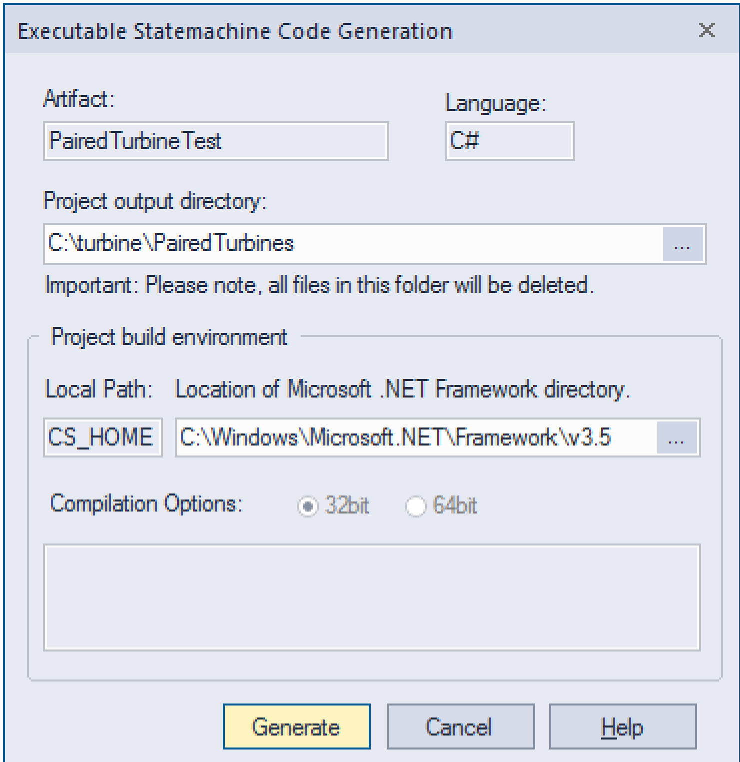
Génération de code (Java sur Linux)



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 d'accès aux fichiers sources s'affichent lorsque le chemin est modifié.
Machine Statemachine exécutable Machine cible	Sélectionnez l'option « À distance ».
Système opérateur	Sélectionnez Linux.
Port	Il s'agit du port de débogage à utiliser. Vous trouverez des références à ce numéro

	de port dans les sections ' Déboguer ' et 'DebugRun' du script d'analyse généré.

Génération de code (autres langages)



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

Dans la dialogue « Génération de code Statemachine Exécutable », les champs « Artefact » et « Langue » affichent le nom de l'élément et la langue de codage tels que définis dans la dialogue « Propriétés » de l'élément.

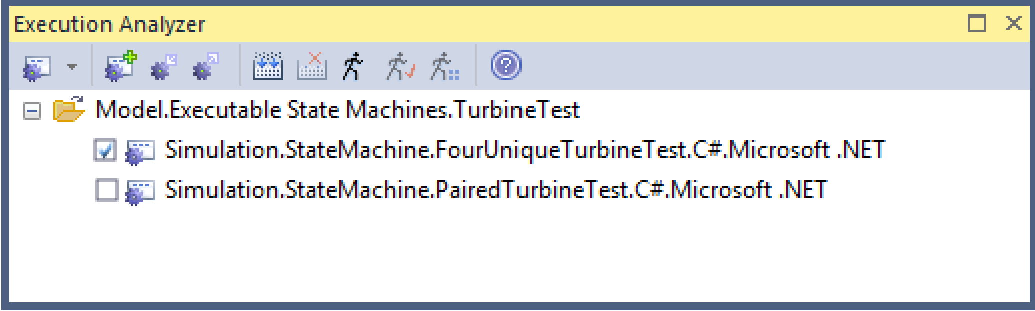
Champ/Option	Description
Répertoire de sortie du projet	Affiche le répertoire dans lequel les fichiers de code générés seront stockés. Si nécessaire, cliquez sur le bouton à droite du champ pour rechercher et sélectionner un autre répertoire.
Environnement de construction du projet	Les champs et informations de ce panneau varient en fonction de la langue définie dans l'élément Artifact et dans le script. Cependant, chaque langue prise en charge fournit une option permettant de définir le chemin d'accès aux frameworks cibles requis pour créer 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 « Génération de code Statemachine Exécutable ».

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 la Statemachine .

ConsoleManager.cs	12/06/2015 10:56 ...	C# Language File	2 KB
ContextManager.cs	12/06/2015 10:56 ...	C# Language File	24 KB
EventProxy.cs	12/06/2015 10:56 ...	C# Language File	2 KB
SimulationManager.cs	12/06/2015 10:56 ...	C# Language File	4 KB

Chaque Statemachine Exécutable généré générera également un script Analyseur d'Exécution , qui est le script de configuration pour la construction, l'exécution et le débogage de l' Statemachine Exécutable .



Code du bâtiment

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 l' Statemachine Exécutable sélectionné, génère à nouveau l'intégralité de la base de code. Le code source est ensuite compilé et la simulation démarrée.
Commande de création de ruban	Compile le code qui a été généré. Cette fonction peut être utilisée 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 de Script	Le script Analyseur d'Exécution généré inclut une commande permettant de construire le code source. Cela signifie que lorsqu'il est actif, vous pouvez le construire directement en utilisant le raccourci intégré Ctrl+Maj+F12.
Générer le résultat	Lors de la construction, toutes les sorties sont affichées sur la page « Build » de la fenêtre Sortie système. Vous pouvez double-cliquer sur les erreurs du compilateur pour ouvrir un éditeur de code source sur la ligne appropriée.

Exploiter le 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 de classe abstrait nommant uniquement les opérations à appeler dans la base de code externe. Vous devez ensuite créer 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.

Débogage de l'exécution des Statemachines Exécutables

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

Débogage d'une Statemachine

Être capable de déboguer un 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 de code source
- Contrôler l'exécution grâce à l'utilisation de points d'action et de points d'arrêt (arrêt en cas d'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, lancez simplement l'exécution de l' Statemachine Exécutable à l'aide du contrôle Simulation . Cependant, selon la nature du comportement à déboguer, nous définirons probablement d'abord des points d'arrêt.

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

Comme tout débogueur, nous pouvons utiliser des points d'arrêt pour examiner la Statemachine en cours d'exécution à un point du code. Localisez une classe d'intérêt dans la fenêtre 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 nommage utilisées pendant la génération. Si vous souhaitez effectuer une pause à 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 l' Statemachine Exécutable , le débogueur s'arrêtera à cette transition et vous pourrez afficher l'état brut des variables pour toutes Statemachines impliquées.

Interrompre l'exécution conditionnellement

Chaque point d'arrêt peut prendre une condition et une instruction trace. Lorsque le point d'arrêt est rencontré et que la condition est évaluée à True, l'exécution s'arrête. Sinon, l'exécution continue 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 éditées.

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 *effet* . Le débogage fournit également fonctionnalités de trace appelées Tracepoints. Il s'agit simplement de points d'arrêt qui, au lieu de s'arrêter, impriment des instructions de trace lorsqu'ils sont rencontrés. La sortie est affichée dans la fenêtre Contrôle Simulation . Ils peuvent être utilisés comme une aide au diagnostic pour afficher et prouver la séquence d'événements et l'ordre dans lequel les instances changent d'état.

Visite 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-la pour déterminer la séquence dans laquelle l'exécution a lieu.

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 aux facilités Simulation pour démontrer visuellement l'exécution en direct de l'artefact Statemachine .

Démarrer une Simulation

La barre d'outils de contrôle Simulation fournit un bouton Rechercher que vous pouvez utiliser pour sélectionner l'artefact Statemachine Exécutable à exécuter . Le contrôle conserve une liste déroulante des Statemachines Exécutables les plus récents parmi lesquels vous pouvez faire votre choix. Vous pouvez également utiliser le menu contextuel d'un artefact Statemachine Exécutable lui-même pour lancer la simulation.

Contrôle de la vitesse

Le contrôle Simulation fournit un paramètre de vitesse. Vous pouvez l'utiliser pour ajuster la vitesse à laquelle la simulation s'exécute. La vitesse est représentée par une valeur 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 d'utiliser les commandes de la barre d'outils pour parcourir manuellement la simulation.

Notation pour States Actif

Au fur et à mesure de l'exécution de l' Statemachine Exécutable , les diagrammes Statemachine concernés s'affichent. L'affichage est mis à jour à la fin de chaque cycle d'étape à étape. Vous remarquerez que seul l' State actif de l'instance qui termine une étape est mis en surbrillance. Les autres States restent grisés.

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

Générer Diagramme de temps

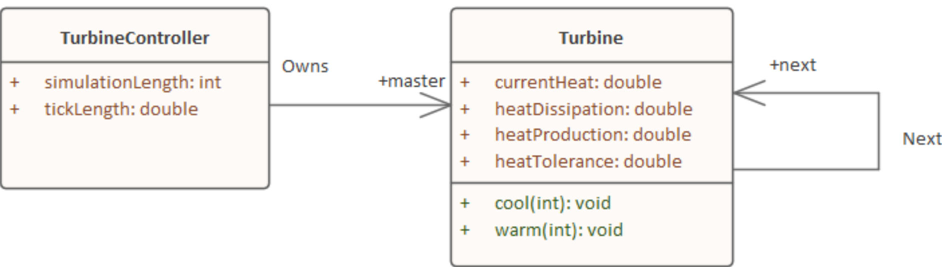
Après avoir terminé la simulation d'un Statemachine Exécutable , vous pouvez générer un diagramme de temps à partir de la sortie. Pour cela :

Dans la barre d'outils de la fenêtre Simulation , cliquez sur « Outils | Générer Diagramme de synchronisation ».

Exemple : Statemachine Exécutable

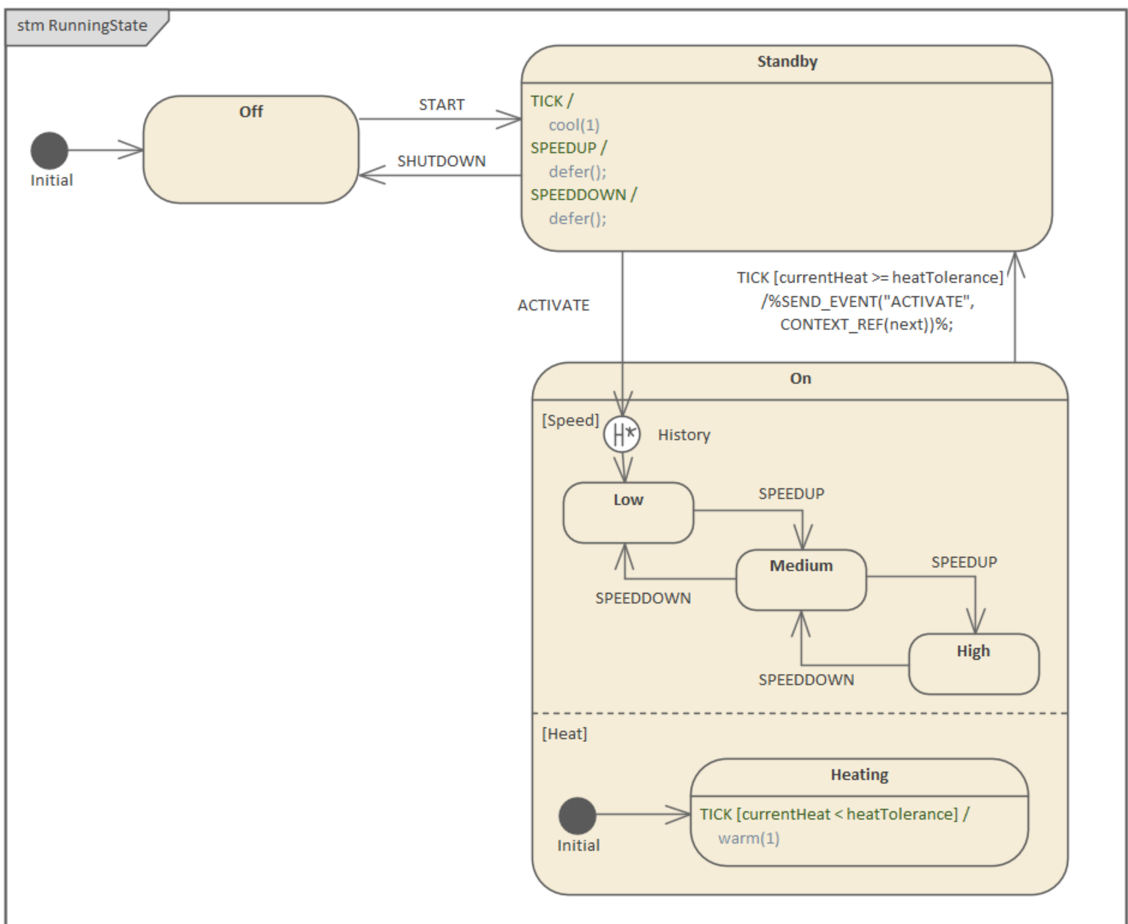
Exemple de classe Modèle

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

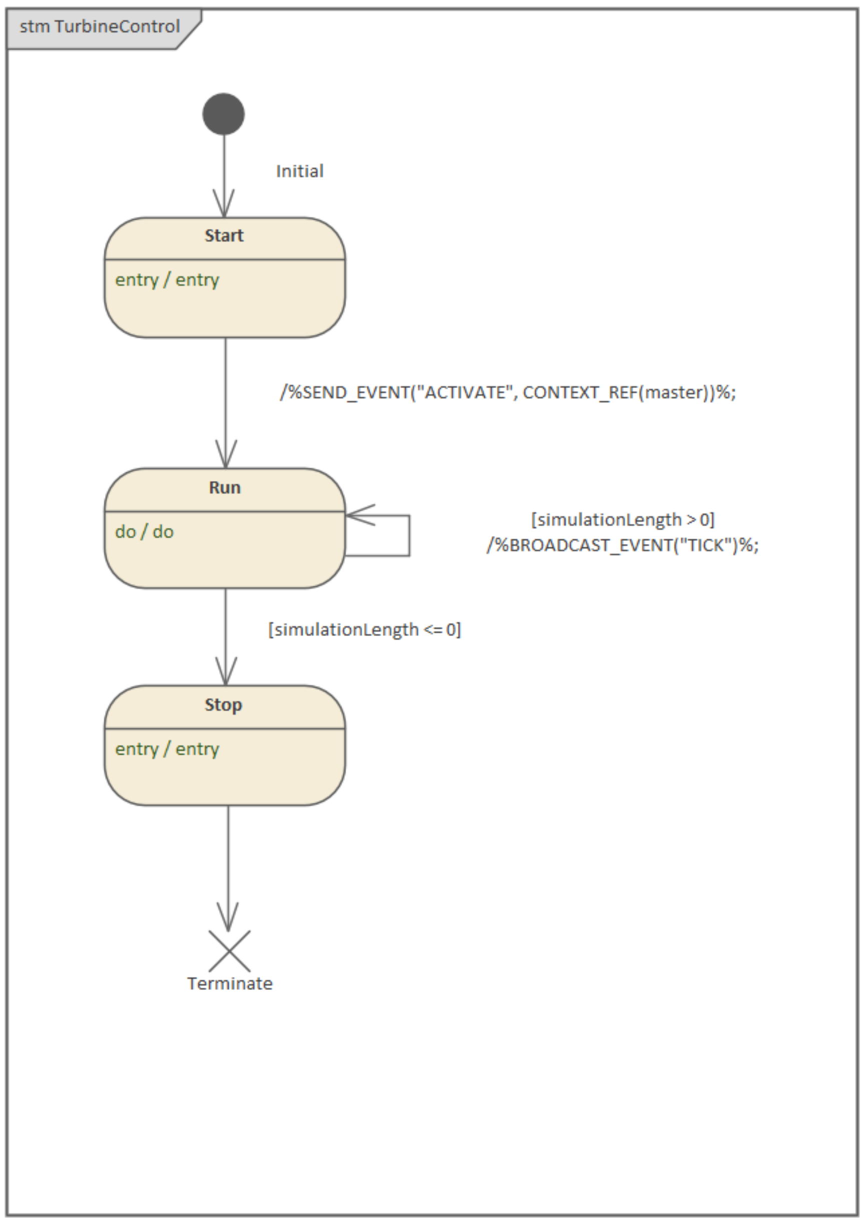


Exemples Statemachines

Ces deux diagrammes montrent les définitions de deux Statemachines . La première référence une autre Statemachine du même type, tandis que la seconde pilote toutes les instances de la première qui existent.

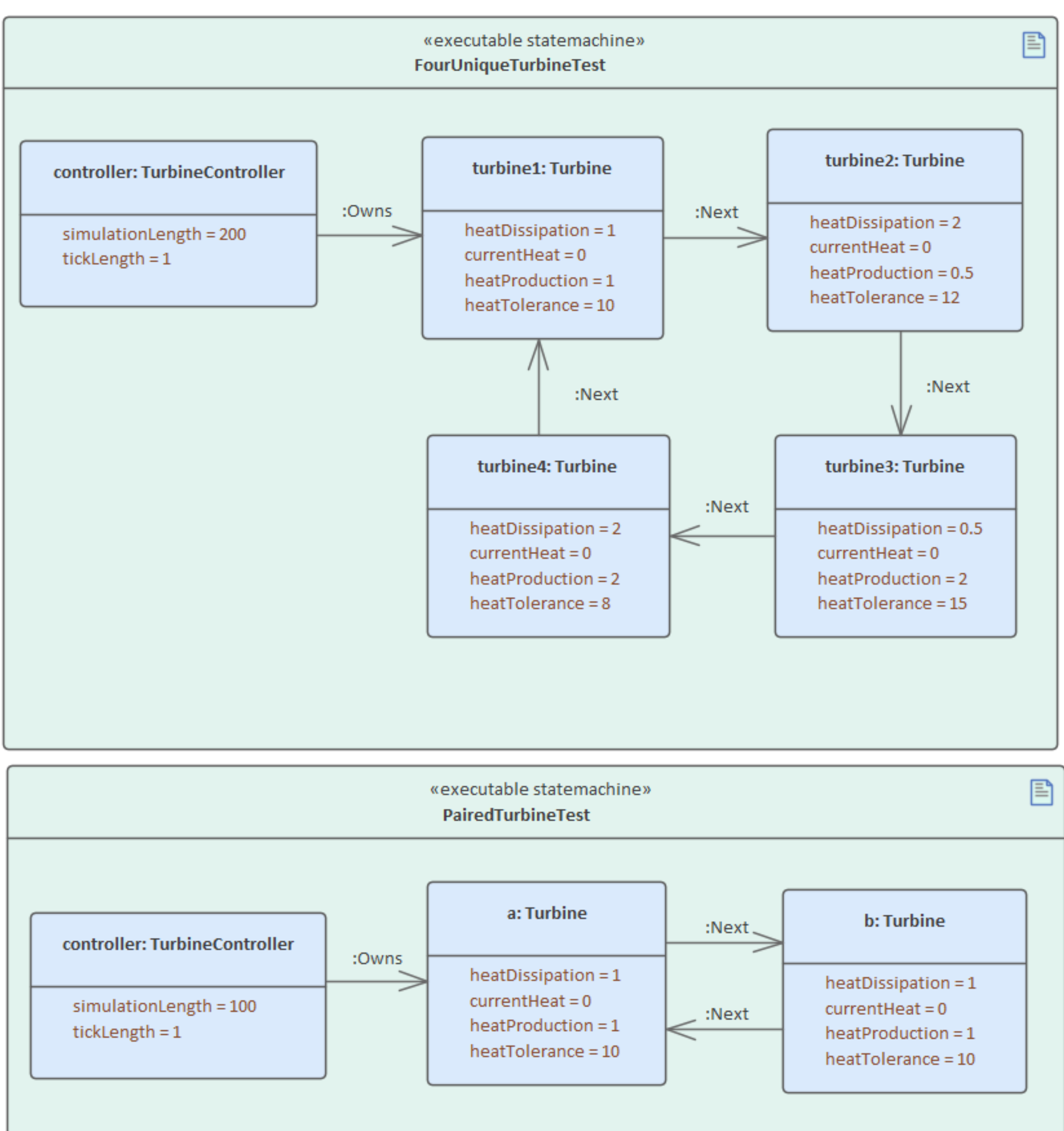


Le contrôleur de niveau supérieur.



Exemples d'artefacts

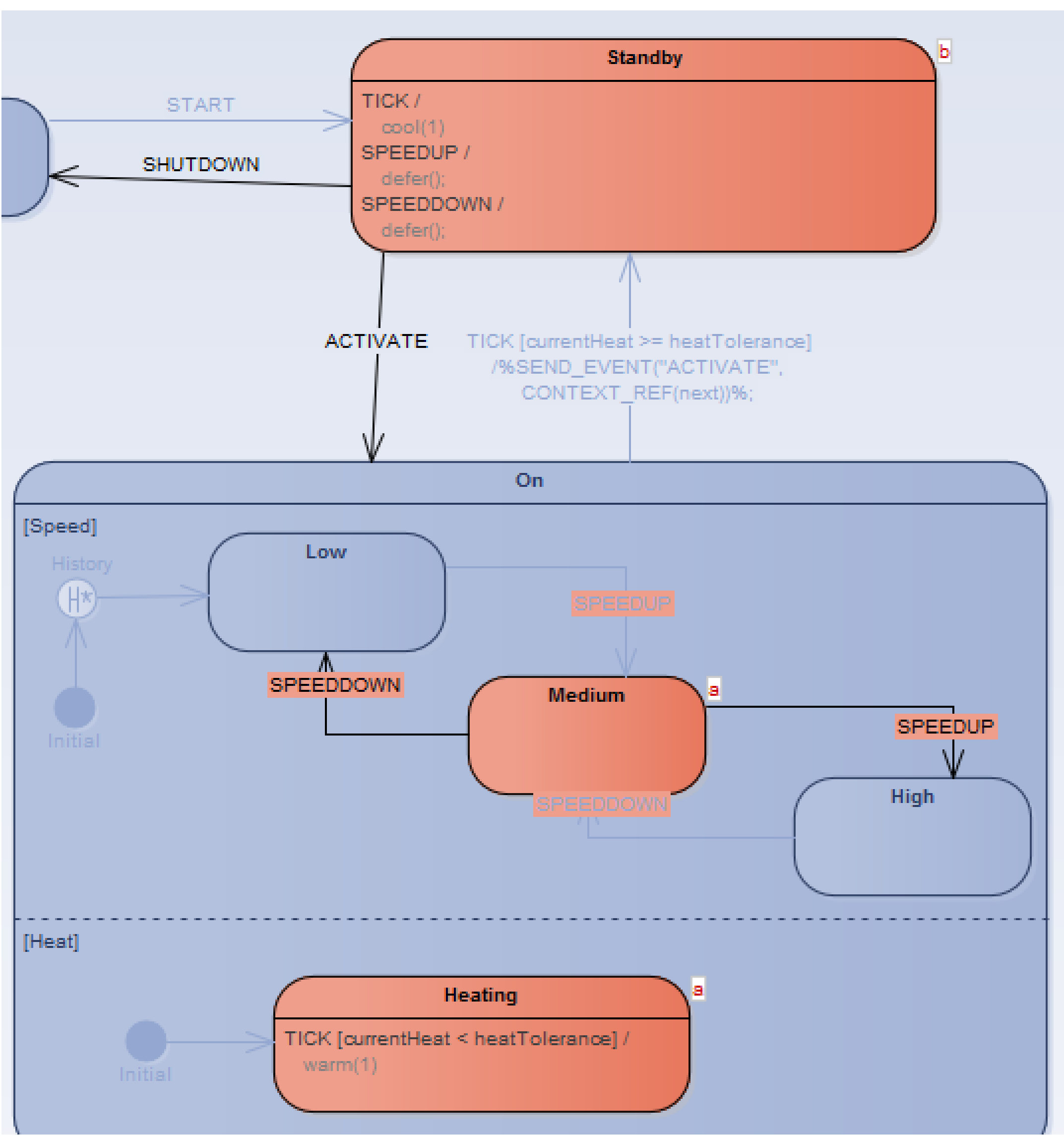
A partir des exemples diagrammes de classes et Statemachine , nous pouvons créer Statemachines Exécutables comme indiqué ici.



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 met en évidence les States actuellement actifs dans toutes Statemachines . Lorsque plusieurs instances d'une Statemachine existent, il affiche également les noms de chaque instance dans cet State .


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

Exemple : Commandes de Simulation

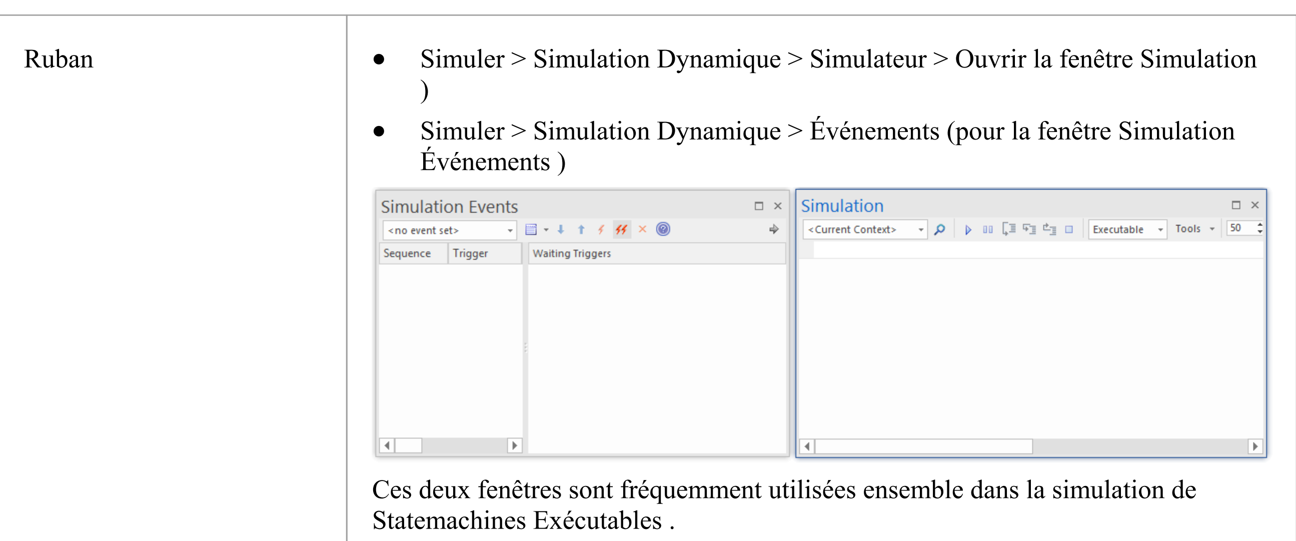
Cet exemple montre comment utiliser la fenêtre Simulation pour observer les messages Trace ou envoyer des commandes pour contrôler une Statemachine . Grâce à cet exemple, vous pouvez examiner :

- Un attribut d'un contexte - la variable membre définie dans la classe, qui est le contexte de la Statemachine ; ces attributs portent des valeurs dans la portée du contexte pour tous les comportements State et les 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 » permet de vider le nombre d'événements actifs de l'état actuel ; elle peut également vider l'événement actuel différé dans le pool

Cet exemple est tiré du modèle EAExample :

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

Accéder

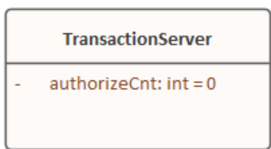


Ruban	• Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation) • Simuler > Simulation Dynamique > Événements (pour la fenêtre Simulation Événements) Ces deux fenêtres sont fréquemment utilisées ensemble dans la simulation de Statemachines Exécutables .

Créer un contexte et Statemachine

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

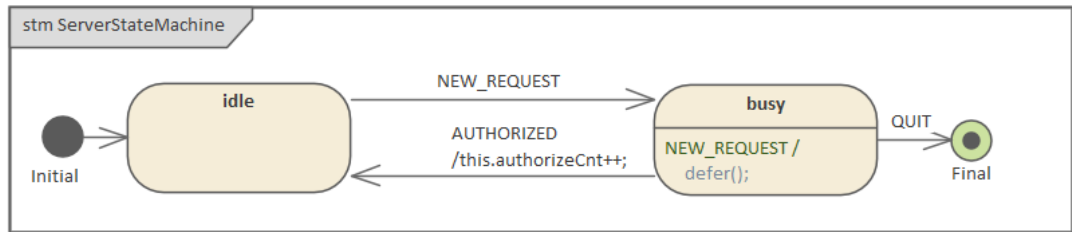
Créer le contexte de la Statemachine



1. Créez un élément de classe 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 | Statemachine '.

Créer la Statemachine



1. Créez un pseudo-état initial appelé *Initial* .
2. Transition vers un State appelé *inactif* .
3. Transition vers un State appelé *busy* , avec le déclencheur NEW_REQUEST.
4. Transition:
   - Vers un pseudo-état Final appelé *Final* , avec le déclencheur QUIT
   - Retour au *repos* , avec le déclencheur AUTORISÉ, avec l'Effet 'this.authorizeCnt++;'

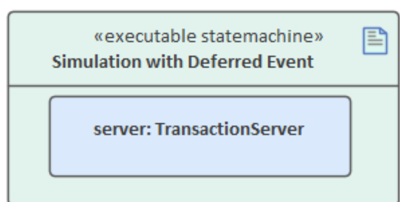
Créer un événement différé pour l' State *occupé*

1. Dessinez une auto-transition pour *être occupé.*
2. Modifiez le « type » de transition en « interne ».
3. Spécifiez le Déclencheur comme étant l'événement que vous souhaitez différer.
4. Dans le champ « Effet », saisissez « 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

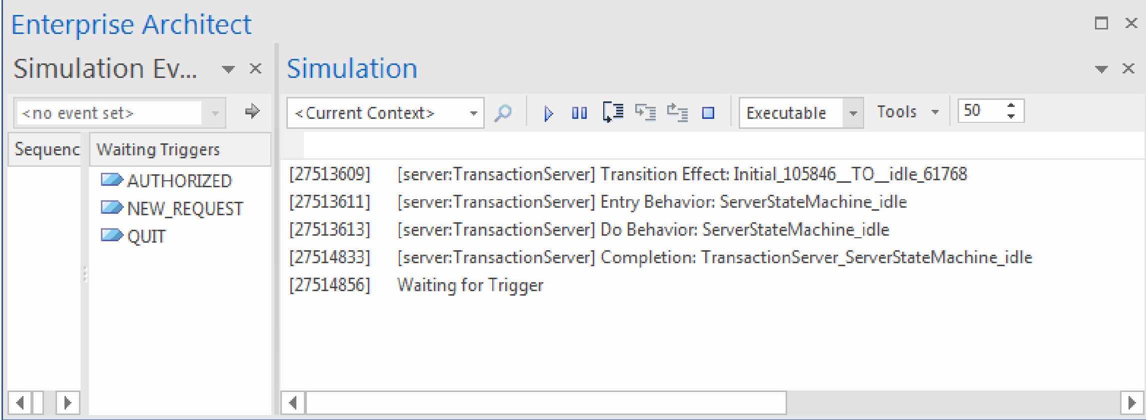


1. Depuis la page « Simulation » de la boîte à outils Diagramme , faites glisser une icône « Statemachine Exécutable » sur le diagramme et appelez l'élément *Simulation avec événement différé.*
2. Ctrl+Glissez l'élément *TransactionServer* depuis la fenêtre Navigateur et déposez-le sur l'Artefact en tant que propriété, avec le *serveur de noms.*
3. Définissez la langue de l'Artefact sur JavaScript , qui ne nécessite pas de compilateur (pour l'exemple ; en production, vous pouvez également utiliser C, C++, C# ou Java, qui support Statemachines Exécutables).
4. Cliquez sur l'artefact et sélectionnez l'option de ruban 'Simulate > States Exécutables > Statemachine > Générer , Build and Exécuter '.

Fenêtre Simulation et commandes

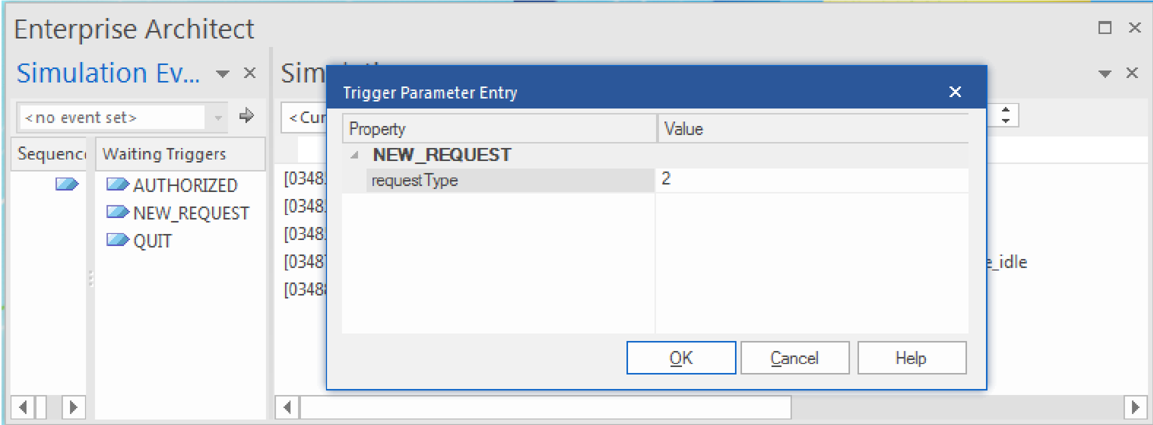
Lorsque la simulation démarre, l'état actuel est *inactif* .

La fenêtre Simulation montre que l'effet de transition, l'entrée et le comportement 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 de Déclencheur NEW_REQUEST, la dialogue « Entrée du paramètre Déclencheur » s'affiche pour prompt des valeurs pour les attributs répertoriés définis dans le signal *RequestSignal* , référencé 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 de 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] [serveur : TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType : 2)

[03611367] [serveur : TransactionServer] Événement envoyé : NEW_REQUEST.RequestSignal(requestType : 2)

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

[03611381] [serveur : TransactionServer] Effet de transition : idle__TO__busy_61772

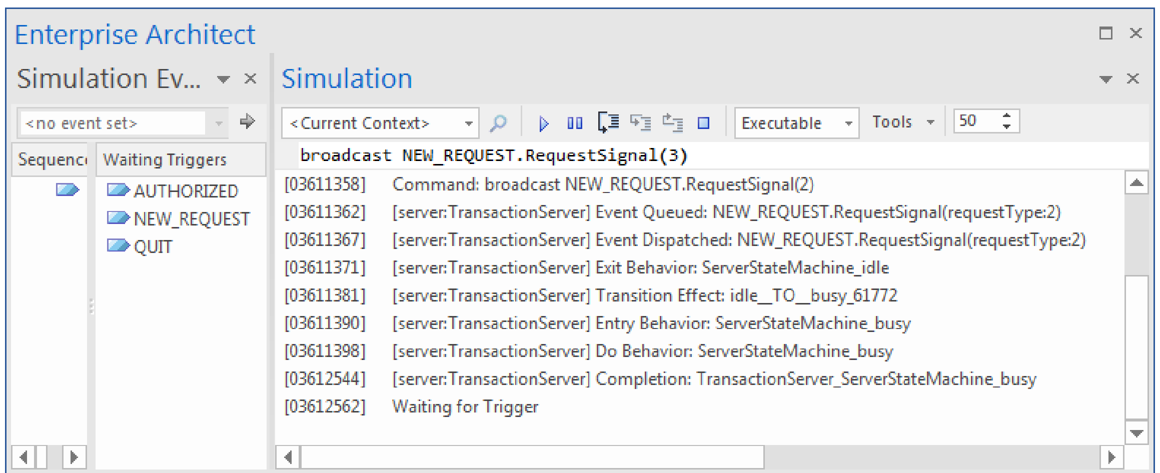
[03611390] [serveur : TransactionServer] Comportement de l'entrée : ServerStateMachine_busy

[03611398] [serveur : TransactionServer] Comportement : ServerStateMachine_busy

[03612544] [serveur : TransactionServer] Achèvement : TransactionServer_ServerStateMachine_busy

[03612562] En attente du Déclencheur

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



[03612562] En attente du Déclencheur

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

[04460233] [serveur : 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 envoyé). Nous pouvons exécuter d'autres commandes à l'aide du champ de texte :

[04655441] En attente du Déclencheur

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

[04664066] [serveur : 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] [serveur : TransactionServer] Événement mis en file d'attente : NEW_REQUEST.RequestSignal(requestType : 5)

[04670312] En attente du Déclencheur

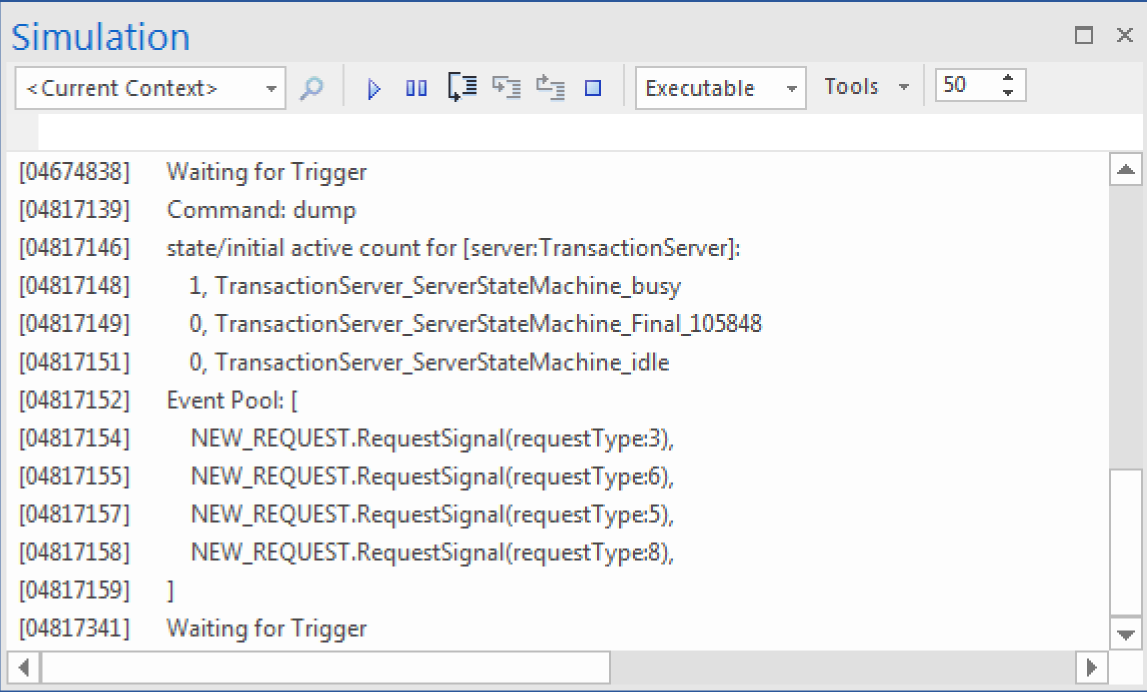
[04674196] Commande : **diffusion NEW_REQUEST.RequestSignal(8)**

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

[04674838] En attente du Déclencheur

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

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



Dans la section « nombre 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 existe quatre occurrences d'événements dans la file d'attente des événements. Chaque instance du signal contient des données différentes.

L'ordre des événements dans le pool est l'ordre dans lequel ils sont diffusés.

eval : Query Exécuter Time Valeur du Contexte

Déclencheur AUTORISÉ,

[04817341] En attente du Déclencheur

[05494672] Commande : diffusion AUTORISÉE

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

[05494680] [serveur : TransactionServer] Événement envoyé : AUTHORIZED

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

[05494686] [serveur : TransactionServer] **Effet de transition : busy__TO__idle_61769**

[05494687] [serveur : TransactionServer] Comportement de l'entrée : ServerStateMachine_idle

[05494688] [serveur : TransactionServer] Comportement : ServerStateMachine_idle

[05495835] [serveur : TransactionServer] Achèvement : TransactionServer_ServerStateMachine_idle

[05495842] [serveur : TransactionServer] **Événement envoyé : NEW_REQUEST.RequestSignal(requestType : 3)**

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

[05495846] [serveur : TransactionServer] Effet de transition : idle__TO__busy_61772

[05495847] [serveur : TransactionServer] Comportement de l'entrée : ServerStateMachine_busy

[05495850] [serveur : TransactionServer] Comportement : ServerStateMachine_busy

[05496349] [serveur : TransactionServer] Achèvement : TransactionServer_ServerStateMachine_busy

[05496367] En attente du Déclencheur

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

[05693348] Pool d'événements : [

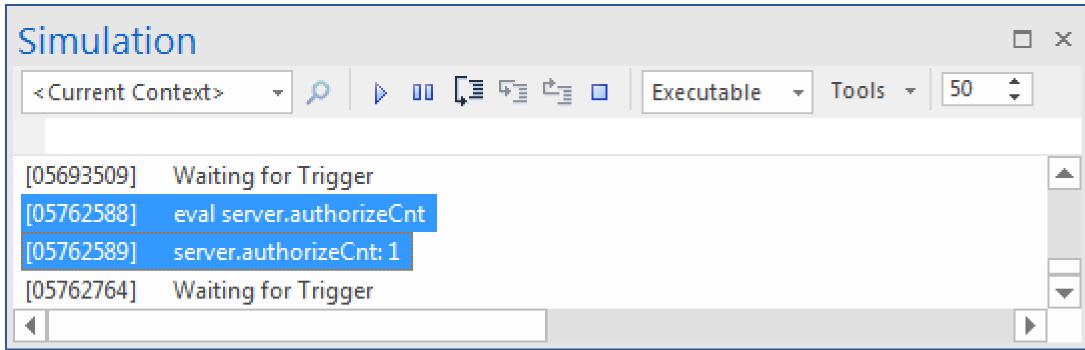
[05693349] NEW_REQUEST.RequestSignal(requestType:6),

[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 AUTHORIZED à nouveau. Lorsque le Statemachine est stable à *busy* , il restera deux événements dans le pool. Exécuter *eval server.suthorizeCnt* à nouveau ; 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 en this->variableName pour C et C++.

Donc, pour toutes les langues, utilisez simplement ce format pour la simulation :

cette.variableName

Exemples :

Dans l'effet de la transition :

ceci.autoriserCnt++;

Dans le comportement d'entrée, de sortie ou de sortie de certains États :

ceci.foo += ceci.bar;

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

this.foo = this.bar + monObjet.iCount + monPointeur->iCount;

sera traduit par :

ceci->foo = ceci->bar + monObjet.iCount + monPointeur->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 démarrer.

- exécuter instance. statemachine
- exécuter tout.tout
- exécuter une instance
- exécuter tous
- exécuter

Par exemple:

exécuter

exécuter tous

exécuter le serveur

exécuter server.myMainStatemachine

diffuser et envoyer l'événement :

- chaîne d'événement de diffusion
- envoyer EventString à l'instance
- envoyer EventString (équivalent à diffuser EventString)

Par exemple:

diffusion Événement1

envoyer l'événement 1 au client

Commande dump :

- décharge
- vidage d'instance

Par exemple:

décharge

serveur de vidage

vider le client

Commande eval :

- évaluer l'instance.variableName

Par exemple:

évaluer client.requestCnt

évaluer le serveur.responseCnt

Commande de sortie :

- sortie

Format de l'EventString :

- EventName.SignalName(liste d'arguments)

Note : la liste des 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 :

- Événement1.Signal1(10, 5) --------- foo = 10; bar = 5
- Event1.Signal1(10,) --------- foo = 10; bar n'est pas défini
- Event1.Signal1(,5) --------- bar = 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 l'EventString comme suit :

- Nom de l'événement

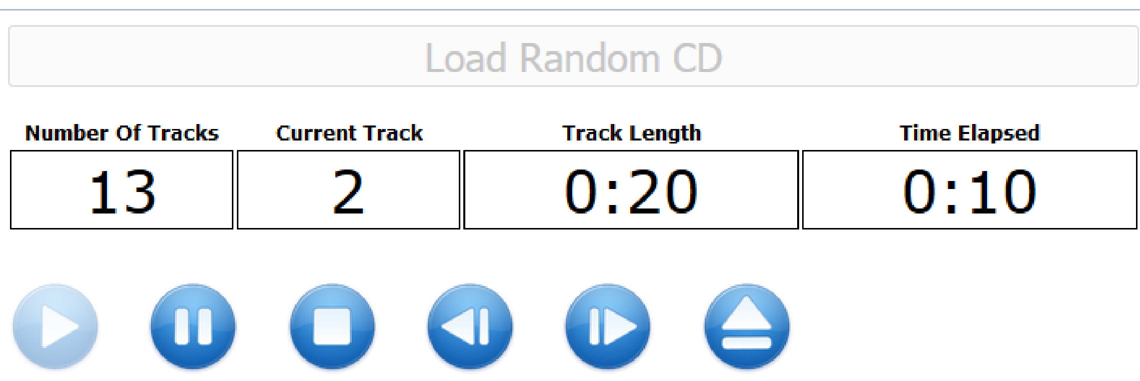
Exemple : Simulation en HTML avec JavaScript

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 *CD Player* et *Regular Expression Parser* , nous allons maintenant vous montrer 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 :

- Basé sur State Actif - le client peut interroger l'état actif actuel, puis « changer » la logique en fonction du résultat de la requête
- Variable d'exécution basée sur - 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 beaucoup de 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 de la piste actuelle.



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

Regular Expression: **(a|b)*abb**

Input string to see if it match: aaababb

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

- CD Player
- Regular Expression Parser

Lecteur CD

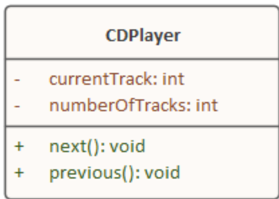
Le comportement d'une application de lecteur CD peut sembler intuitif ; cependant, il existe de nombreuses règles liées au moment où les boutons sont activés et désactivés, à ce qui est affiché dans les champs de texte de la fenêtre et à ce qui se passe lorsque vous fournissez des événements à l'application.

Supposons que notre exemple de lecteur CD possède ces fonctionnalités :

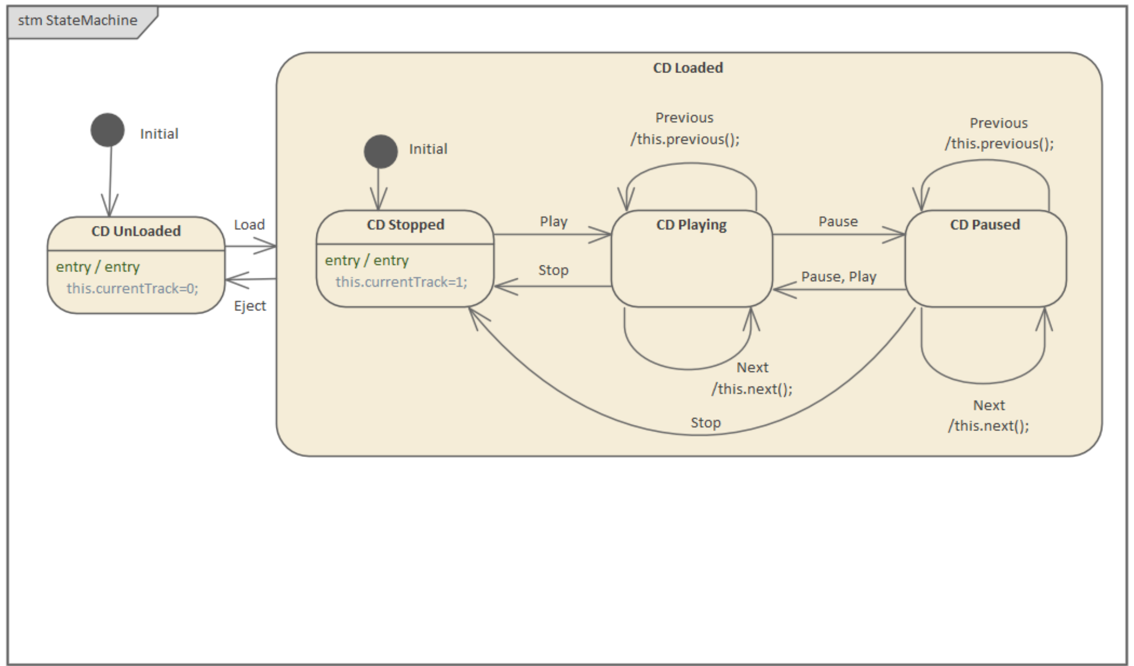
- Boutons - Charger un CD aléatoire, Lecture, Pause, Arrêt, Piste précédente, Piste suivante et Éjecter
- Affichages - Nombre de pistes, piste actuelle, durée de la piste et temps écoulé

Statemachine pour lecteur CD

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



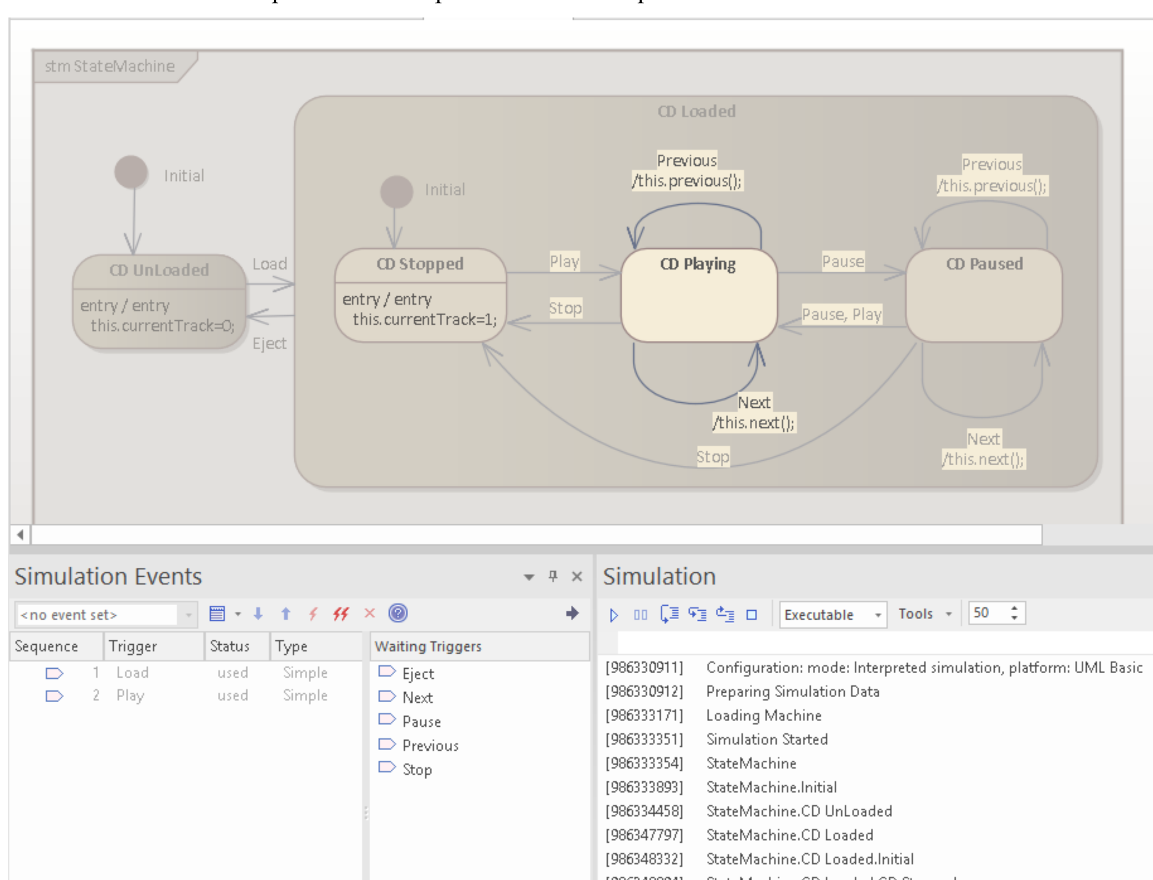
Une Statemachine est utilisée pour décrire les états du lecteur CD :



- Au niveau supérieur, la Statemachine a deux States : *CD non chargé* et *CD chargé*
- *Le CD chargé* peut être composé de trois States simples : *CD arrêté* , *CD en lecture* , *CD en pause*
- Les transitions sont définies avec déclencheurs pour les événements Load, Eject, Play, Pause, Stop, Previous et Next
- Les comportements State et les effets de transition sont définis pour modifier la valeur des attributs définis dans *CDPlayer* ; par exemple, l'événement 'Précédent' déclencheur l'auto-transition (si l'état actuel est *CD en lecture* ou *CD en pause*) 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 typant *CDPlayer* , puis simuler le

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 ainsi que ses attributs et ses 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 qui est indépendant des contextes réels, qui sont définis dans la généralisation du *ContextManager* , comme *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* typé dans la classe *Arbitre,* et deux joueurs - *joueurA* et *joueurB* - typés dans la classe *Joueur.* La classe *Arbitre* et la classe *Joueur* définiront chacune leur(s) propre(s) Statemachine (s).

/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 de signal, qui spécifie une classe de signal ; nous définissons ensuite des attributs pour la classe de signal. Chaque occurrence d'événement possède une instance du signal, contenant 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 tard dans ce sujet.
/html/index.html	Ce fichier définit l' Interface Utilisateur , comme les boutons et les entrées permettant de déclencher Événements ou d'afficher des informations. Vous pouvez définir du CSS et JavaScript dans ce fichier.

Personnaliser index.html et statemachineGUI.js

Apportez ces modifications aux fichiers générés :

- Créer des boutons et des affichages
- Créer 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, définissez-la sur Next Track lorsque le temps écoulé atteint la longueur de la piste
- Créer JavaScript comme gestionnaire d'événements du bouton « onclick »
- Une fois qu'un événement est diffusé, demandez l' State actif et valeur d'exécution pour *cdPlayer.currentTrack*
- Lors de 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 sa longueur

L'exemple complet

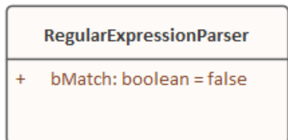
L'exemple est accessible depuis la page « Ressources » du site Web Sparx Systems , en cliquant sur ce lien :
CD Player Simulation

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

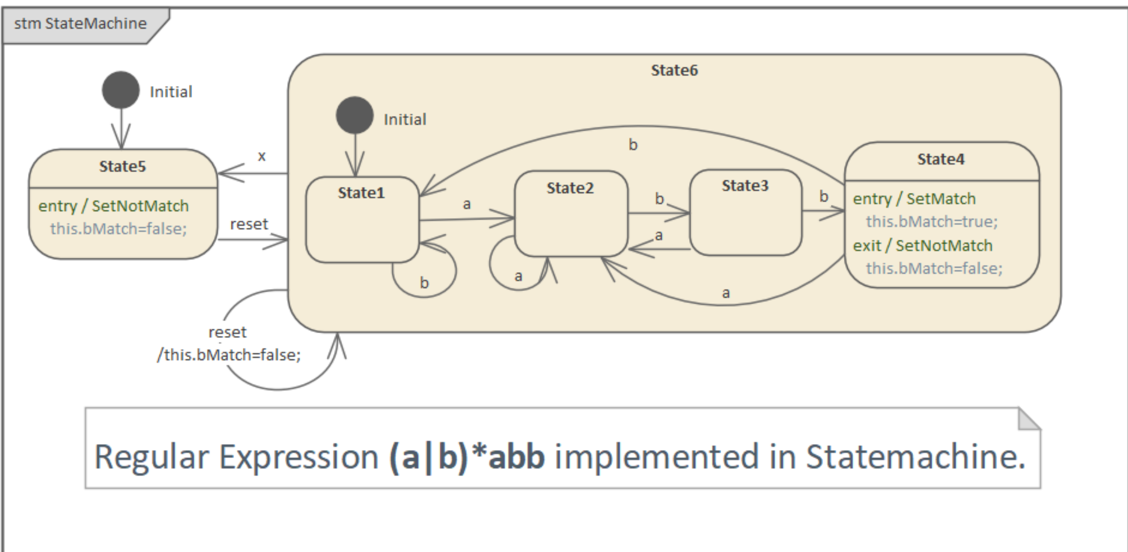
Parser d'expressions régulières

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 des événements *a* , *b* , *x* et *reset*
- À l'entrée de l'État 4, bMatch est défini sur True ; à la sortie de l'État 4, bMatch est défini sur False
- À l'entrée de State5, bMatch est défini sur False
- Lors de l'auto-transition de l'état 6, 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 comme 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
- Lors de 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 Web Sparx Systems , en cliquant sur ce lien :

Regular Expression Parser Simulation

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 (s'il est défini) à l'entrée, mais seulement après la fin de tout comportement d'effet associé à la transition entrante. De plus, si un comportement doActivity est défini pour l' State , ce comportement commence son exécution immédiatement après l'exécution du comportement d'entrée.

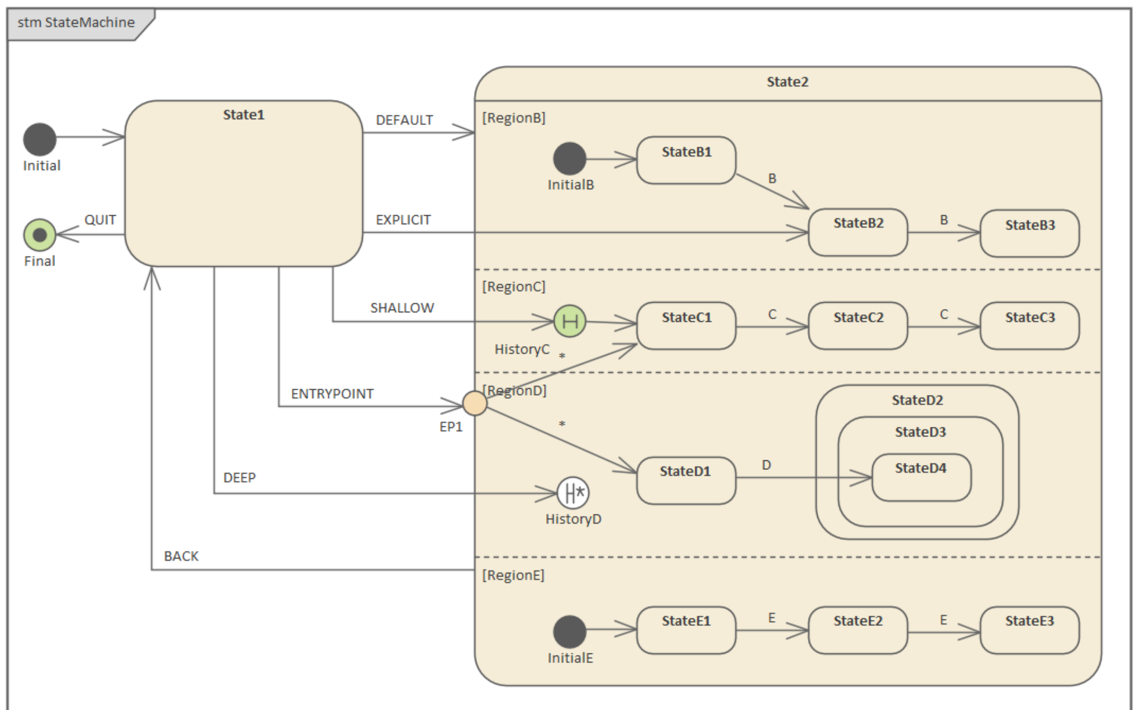
Pour un State composite avec une ou plusieurs régions définies, un certain nombre d'alternatives existent pour chaque région :

- *Entrée par défaut* : Cette situation se produit lorsque l' State composite propriétaire est la cible directe d'une transition ; après avoir exécuté le comportement d'entrée et avoir forgé une éventuelle exécution de comportement doActivity, l'entrée State continue à partir d'un pseudo-état initial via sa transition sortante (appelée 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 dans l' State composite propriétaire, alors ce sous-état devient actif et son comportement d'entrée est exécuté après l'exécution du comportement d'entrée de l' State composite contenant
  Cette règle s'applique de manière récursive si la transition se termine sur un sous-état indirect (profondément imbriqué)
- *Entrée d'historique peu profonde* : si la transition entrante se termine sur un pseudo-état d'historique peu profonde de cette région, le sous-état actif devient le sous-état qui était le plus récemment actif (sauf l'état final) avant cette entrée, à moins qu'il ne s'agisse de la première entrée dans cet State ; s'il s'agit de la première entrée dans cet State ou si l'entrée précédente avait atteint un état final, une transition d'historique peu profonde 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 pseudo-état cible est de type deepHistory et que la règle est appliquée de manière récursive à tous les niveaux de la configuration State active en dessous de celui-ci
- *Entrée au point d'entrée* : si une transition entre dans l' State composite propriétaire via un pseudo-état entryPoint, alors la transition sortante provenant du point d'entrée et pénétrant dans l' State de cette région est prise ; s'il y a plus de transitions sortantes à partir des points d'entrée, chaque transition doit cibler une région différente et toutes les régions sont activées simultanément

Pour States orthogonaux à plusieurs Régions, si la Transition entre explicitement dans une ou plusieurs Régions (dans le cas d'une Fourche ou d'un point d'entrée), ces Régions sont saisies 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 à la Statemachine .
2. Cliquez-droit sur *MyClass* dans la fenêtre Navigateur et sélectionnez l'option 'Ajouter | 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 *l'État 2* sur le diagramme , cliquez-droit dessus et sélectionnez l'option « Avancé | Définir les sous-états simultanés », puis définissez *la Région B, la Région C, la Région D* et *la Région E.*
3. Cliquez-droit sur *State2* et sélectionnez l'option 'Nouvel élément enfant | Point d'entrée' pour créer le point d'entrée *EP1* .
4. Dans *la RégionB* , créez les éléments *InitialB* , transition vers *l'ÉtatB1* , transition vers *l'ÉtatB2* , transition vers *l'ÉtatB3* ; 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 to *StateC1* , transition to *StateC2* , transition to *StateC3* ; toutes les transitions déclenchées par Event *C.*
6. Dans *RegionD* , créer les éléments deep *HistoryD* (cliquez-droit sur le noeud History | Advanced | Deep History | check), transition vers *StateD1* , créer *StateD2* comme parent de *StateD3* , qui est parent de *StateD4* ; transition de *StateD1* à *StateD4* ; déclenchée 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 les transitions du point d'entrée *EP1* vers *l'État C1* et *l'État D1.*

Dessiner des transitions pour différents types d'entrées :

1. Entrée par défaut : *État 1* à *État 2* ; déclenché par l'événement DEFAULT.
2. Entrée explicite : *État1* à *ÉtatB2* ; déclenché par l'événement EXPLICIT.
3. Entrée d'historique superficiel : *État 1* vers *Historique C* ; déclenchée par l'événement SHALLOW.
4. Entrée d'historique profond : *État 1* à *Historique D* ; déclenché par l'événement DEEP.
5. Point d'entrée Entrée : *État 1* à *EP1* ; déclenché par l'événement ENTRYPOINT.

Autres transitions :

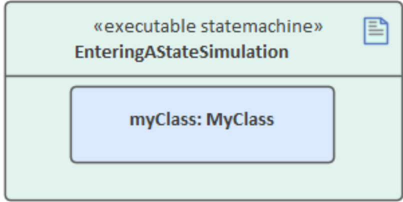
1. Sortie State composite : de *l'État 2* à *l'État 1* ; déclenchée par l'événement BACK.
2. *État 1* à *Final* , déclenché par l'événement QUIT.

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.)

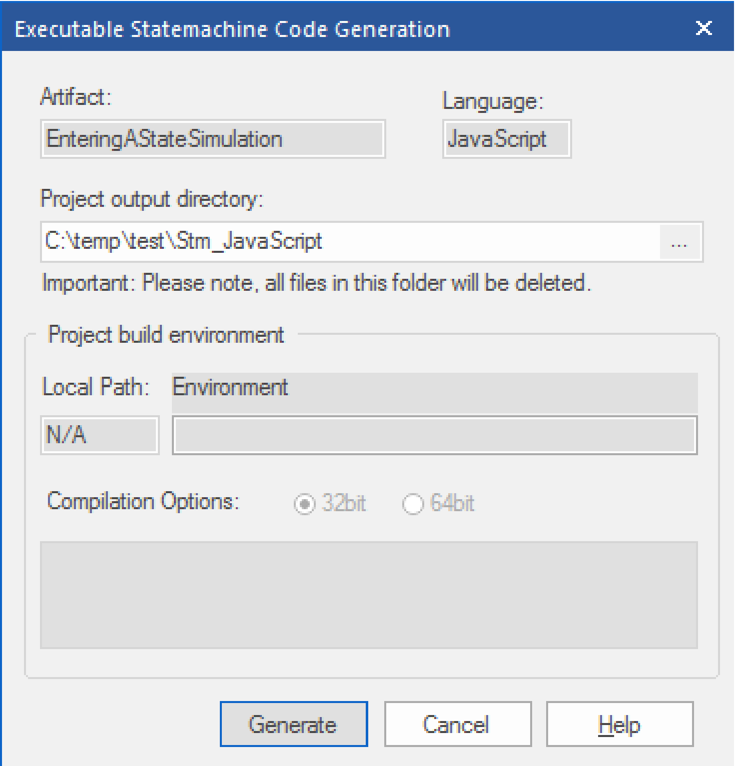
1. Sur la page « Simulation » de la boîte à outils Diagramme , faites glisser l'icône Statemachine Exécutable sur un diagramme et créez un artefact nommé *EnteringAStateSimulation* . Définissez le langage sur JavaScript .
2. Maintenez Ctrl+faites glisser l'élément *MyClass* de la fenêtre Navigateur sur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



Génération de code

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

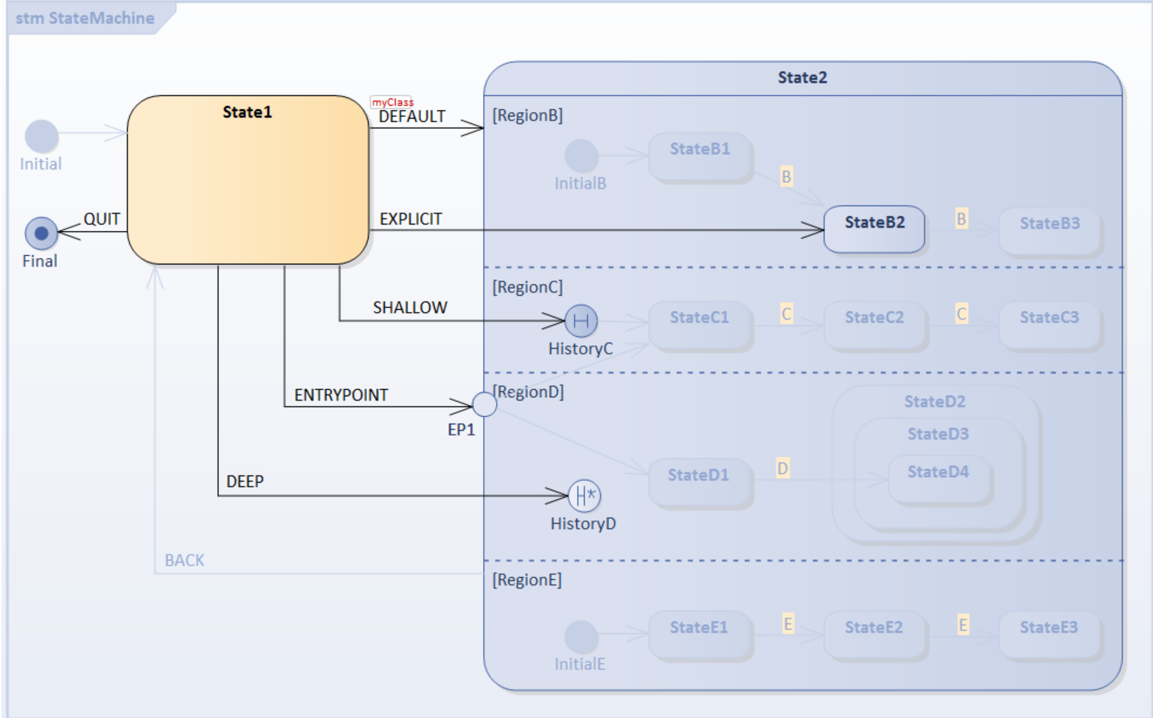
Note : le contenu de ce répertoire sera effacé avant la génération ; assurez-vous de spécifier un répertoire utilisé uniquement à des fins de simulation Statemachine .



Exécuter Simulation

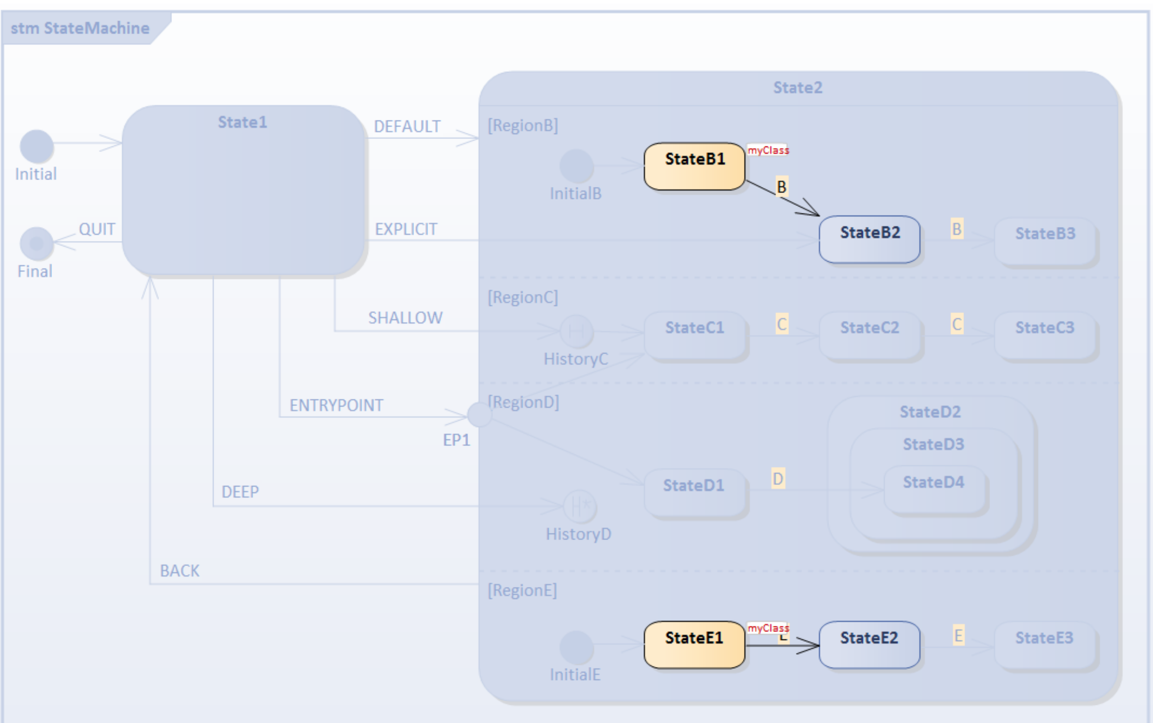
Conseils : Vous pouvez visualiser la séquence de trace d'exécution à partir de la fenêtre Simulation , que vous ouvrez en sélectionnant l'option de ruban 'Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation '

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].

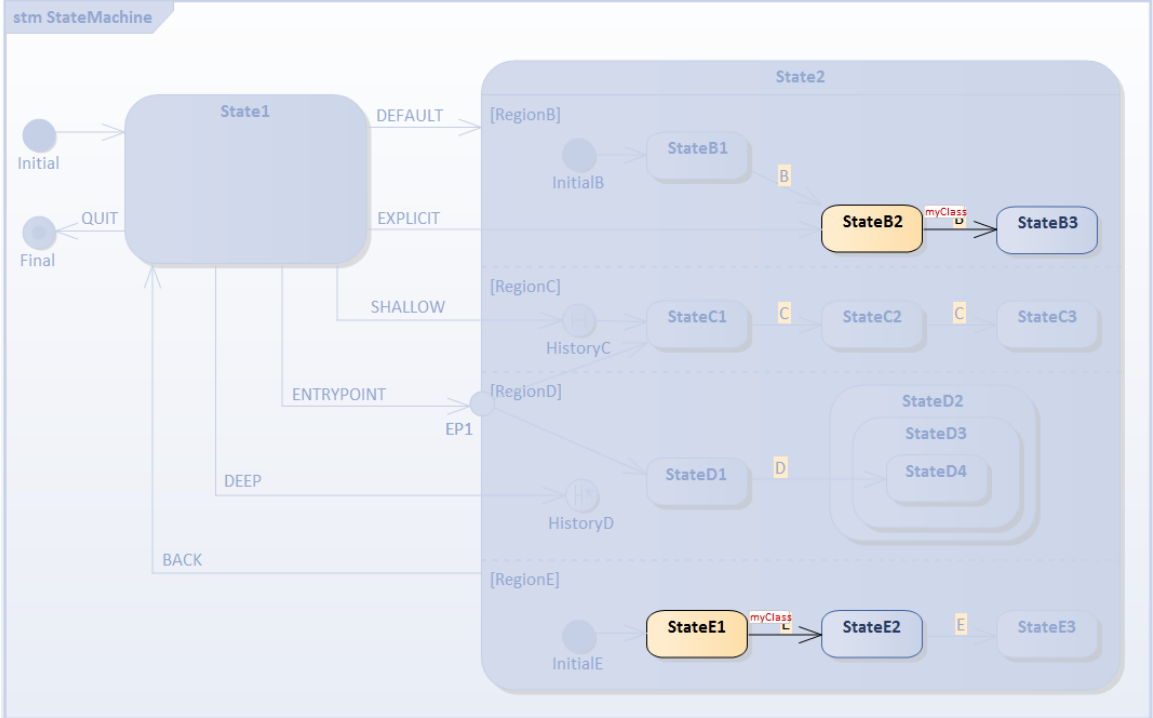


- *La région B* est activée car elle définit *InitialB* ; la transition sortant de celle-ci sera exécutée, *l'état B1* est l'état actif

- *RegionE* est activé car il définit *InitialE* ; la transition sortant de celui-ci sera exécutée, *StateE1* est l'état actif
- *Les régions C* et *D* sont inactives car aucun pseudo-état initial n'a été défini

Sélectionnez le Déclencheur [BACK] pour réinitialiser.

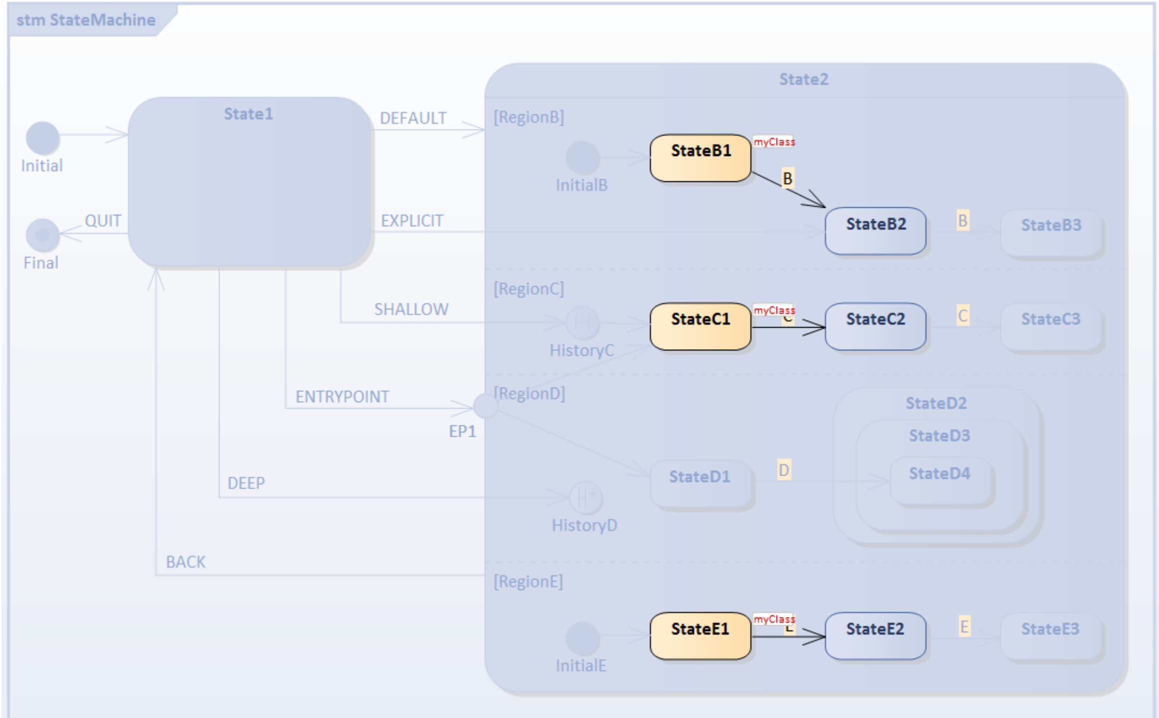
2) Sélectionnez l'entrée explicite : Déclencheur Séquence [EXPLICIT].



- *La région B* est activée car la transition cible le sommet contenu *StateB2*
- *RegionE* est activé car il définit *InitialE* ; la transition sortant de celui-ci sera exécutée, *StateE1* est l'état actif
- *Les régions C* et *D* sont inactives car aucun pseudo-état initial n'a été défini

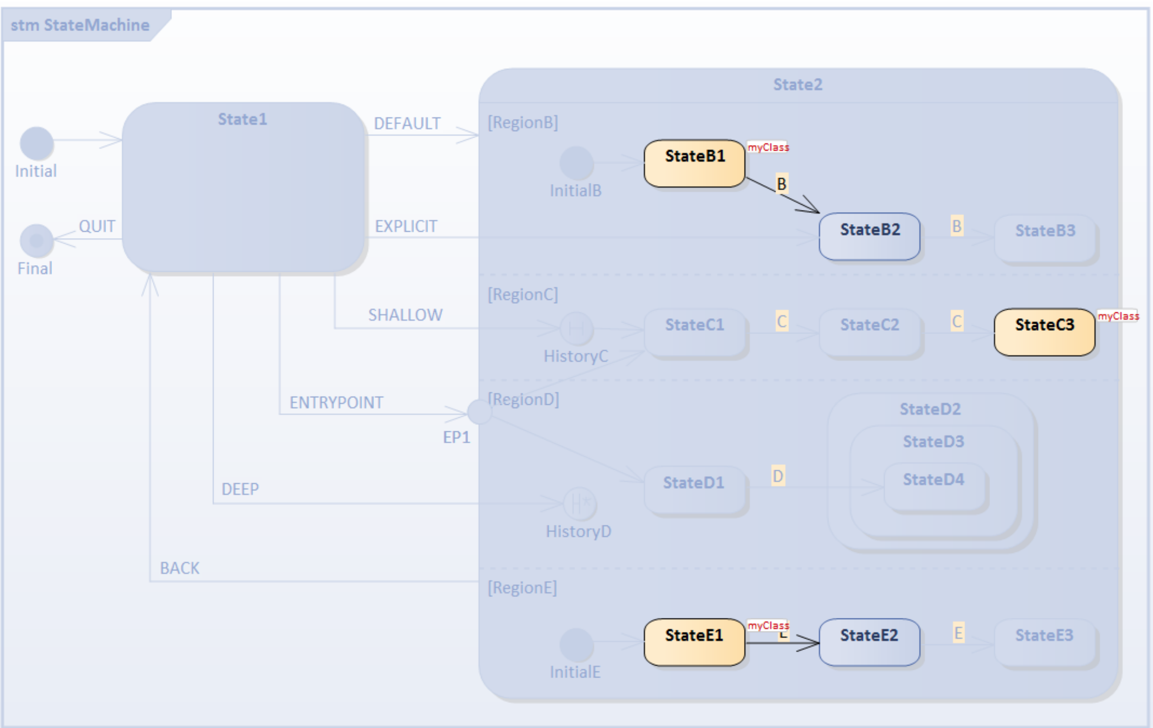
Sélectionnez le Déclencheur [BACK] pour réinitialiser.

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



- *La région C* est activée 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 « mémoriser »), la transition sortant de *HistoryC* vers *StateC1* est exécutée
- *La région B* est activée car elle définit *InitialB* ; la transition sortant de celle-ci sera exécutée, *l'état B1* est l'état actif
- *RegionE* est activé car il définit *InitialE* ; la transition sortant de celui-ci sera exécutée, *StateE1* est l'état actif
- *La région D* est inactive car aucun pseudo-état initial n'a été défini

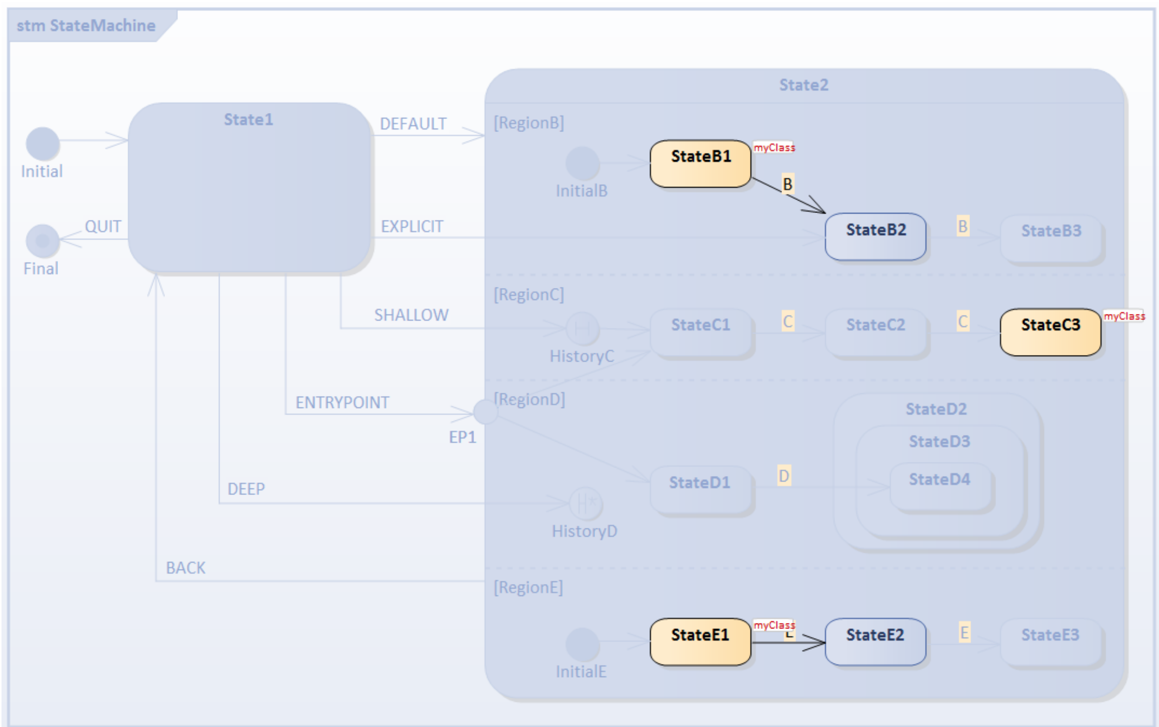
4) Préparez-vous au test de l'entrée d'historique superficielle : Séquence Déclencheur [C, C].



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

Sélectionnez le Déclencheur [BACK] pour réinitialiser.

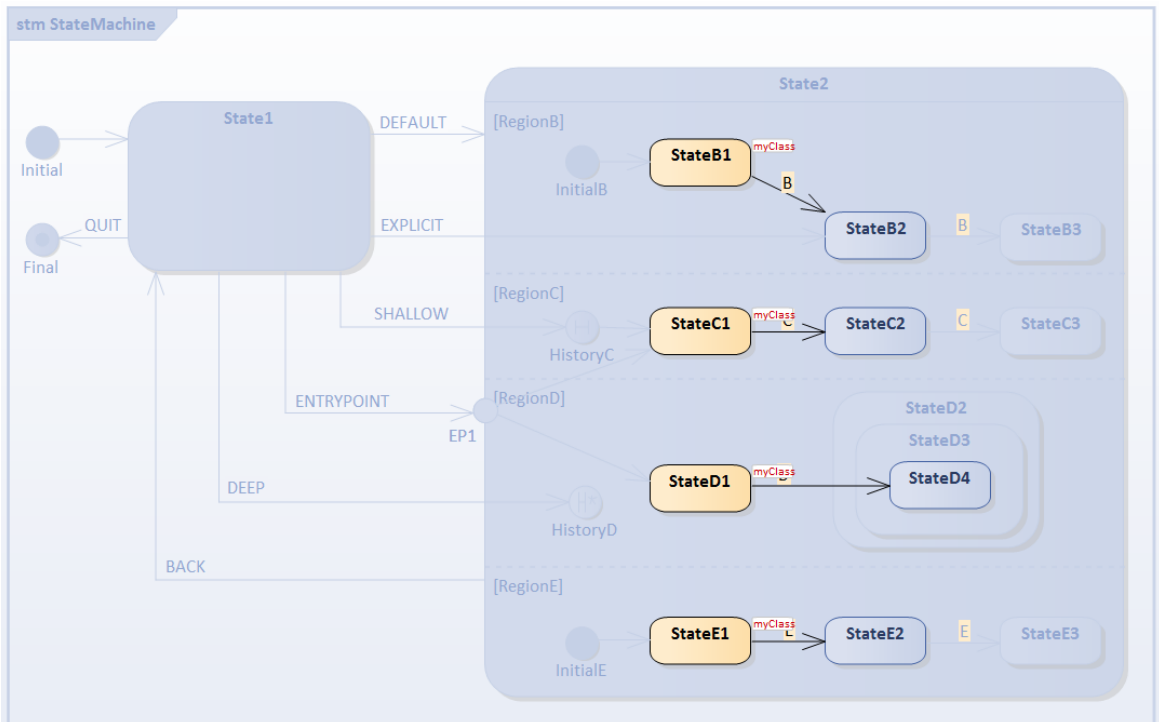
5) Sélectionnez l'entrée d'historique peu profonde : Déclencheur Séquence [SHALLOW].



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

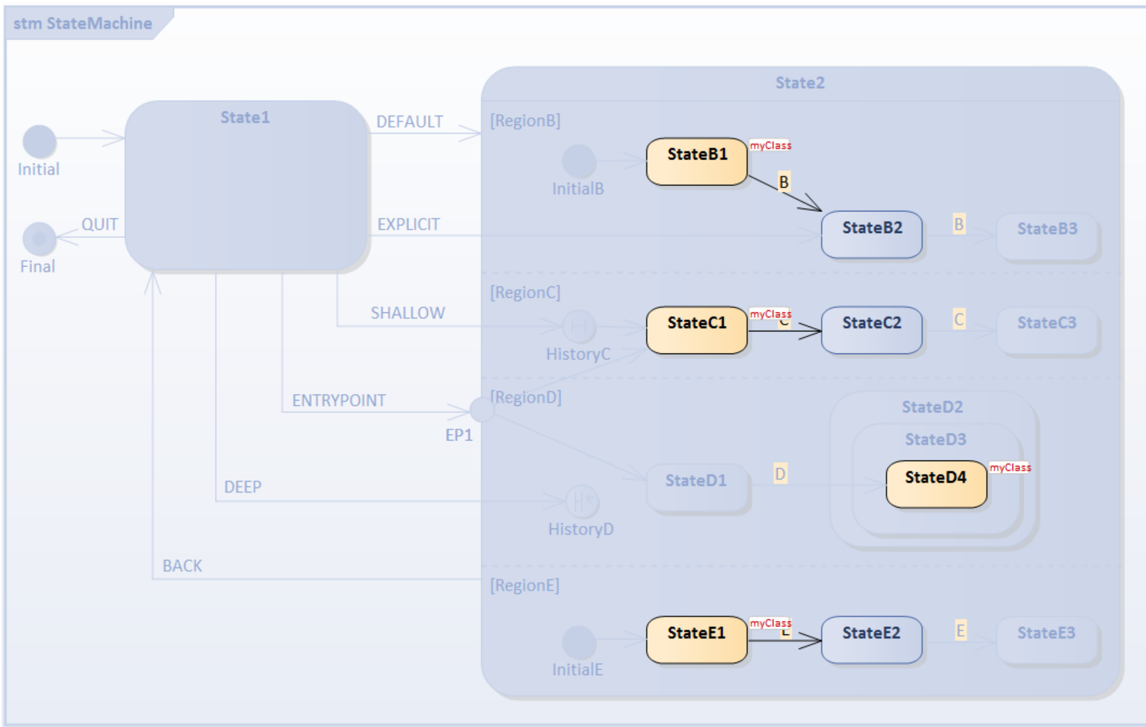
Sélectionnez le Déclencheur [BACK] pour réinitialiser.

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



- *La région C* est activée car la transition depuis *EP1* cible l' *État C1* contenu
- *La région D* est activée car la transition depuis *EP1* cible l' *État D1* contenu
- *La région B* est activée car elle définit *InitialB* ; la transition sortant de celle-ci sera exécutée, *l'état B1* est l'état actif
- *RegionE* est activé car il définit *InitialE* ; la transition sortant de celui-ci sera exécutée, *StateE1* est l'état actif

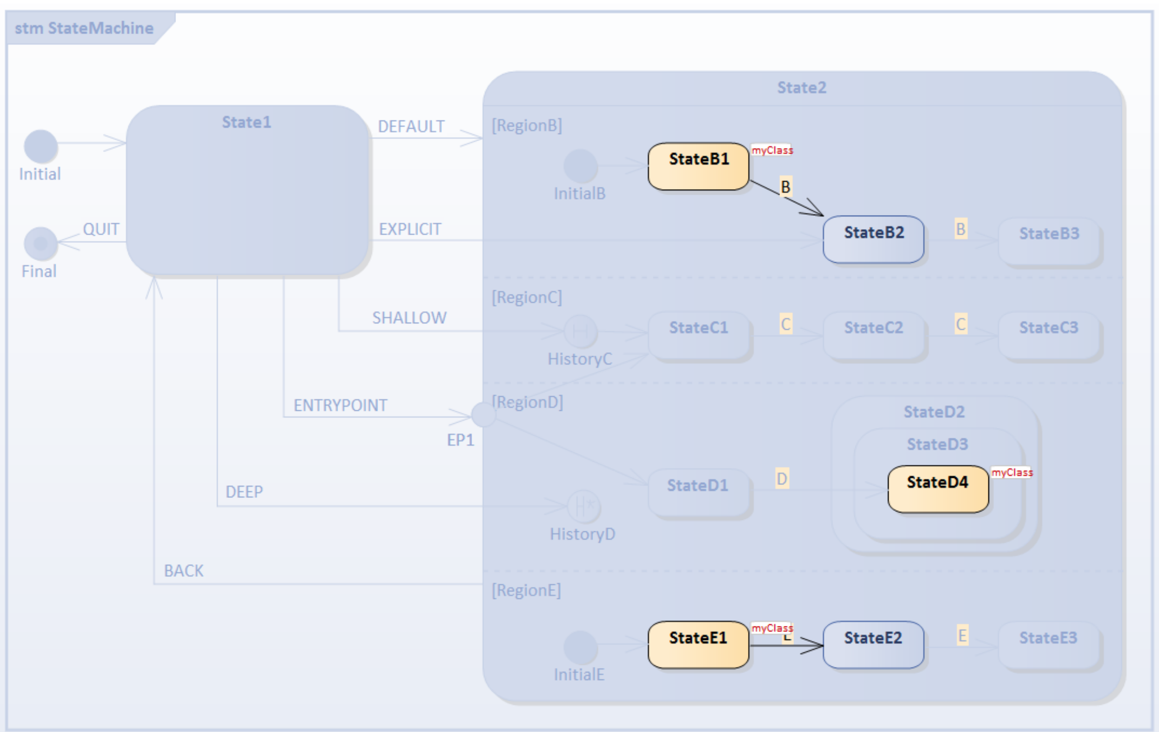
7) Préparez-vous à tester Deep History : Déclencheur Séquence [D].



- 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 SORTIE

- myClass[MyClass].State1__TO__HistoryD_105793_61752 Effet

- maClasse[MaClasse].StateMachine_State2 ENTRÉE

- maClasse[MaClasse].StateMachine_State2 FAIRE

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

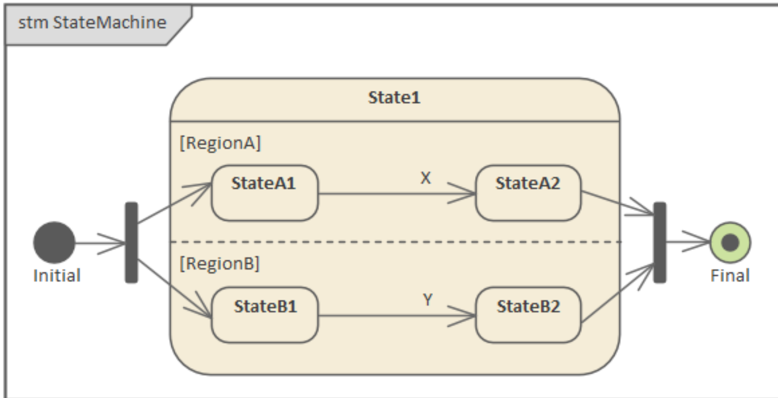
Exemple : Fourche et Joindre

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 de fourche 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 exécutent une fonction de synchronisation, selon laquelle toutes les transitions entrantes doivent être terminées avant que l'exécution puisse se poursuivre via une transition sortante.

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

Modélisation Statemachine



Contexte de Statemachine

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

Statemachine

- Ajoutez un nœud *initial* , une *fourche* , un State nommé *State1* , une *jointure* et un *final* au diagramme
- Agrandir *l'état1* , cliquez-droit dessus sur le diagramme et sélectionnez l'option « Avancé | Définir les sous-états simultanés | Définir » et définissez *la région A* et *la région B*
- Dans *RegionA* , définissez *StateA1* , transition vers *StateA2* , déclenchée par l'événement *X*
- Dans *la région B* , définissez *l'État B1* , transition vers *l'État B2* , déclenchée par l'événement *Y*
- Dessiner d'autres transitions : *Initial* vers *Fork* ; *Fork* vers *StateA1* et *StateB1* ; *StateA2 et StateB2* vers *Join* ; *Join* vers *Final*

Simulation

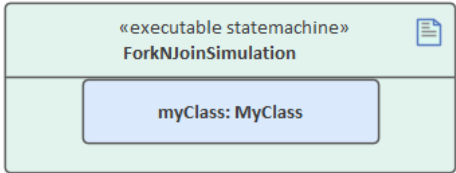
Artefact

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).

- Depuis 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 ».

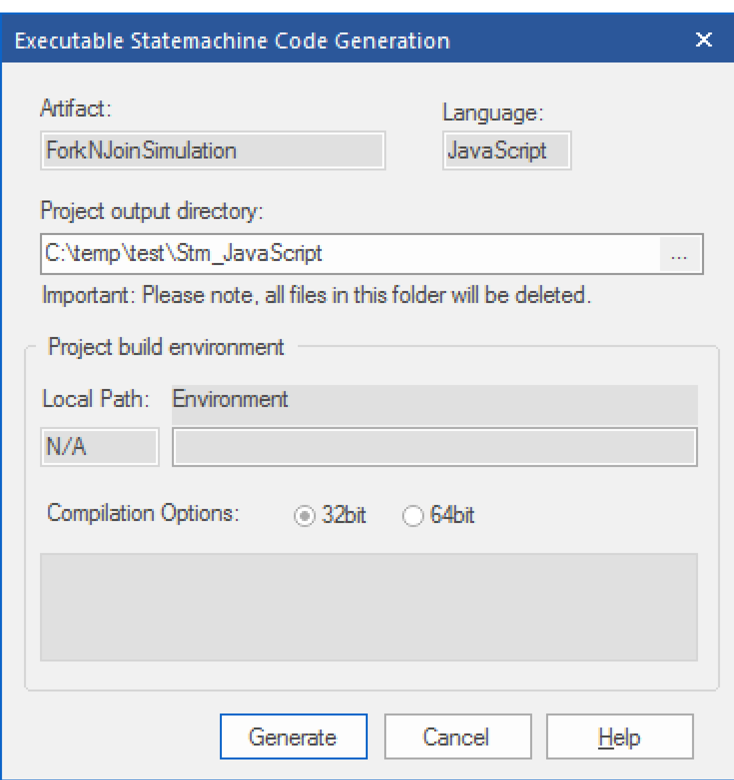
- Ctrl+Drag *MyClass* depuis la fenêtre Navigateur et déposez-le sur l'artefact *ForkNJoinSimulation* en tant que propriété ; donnez-lui le nom *myClass*



Génération de code

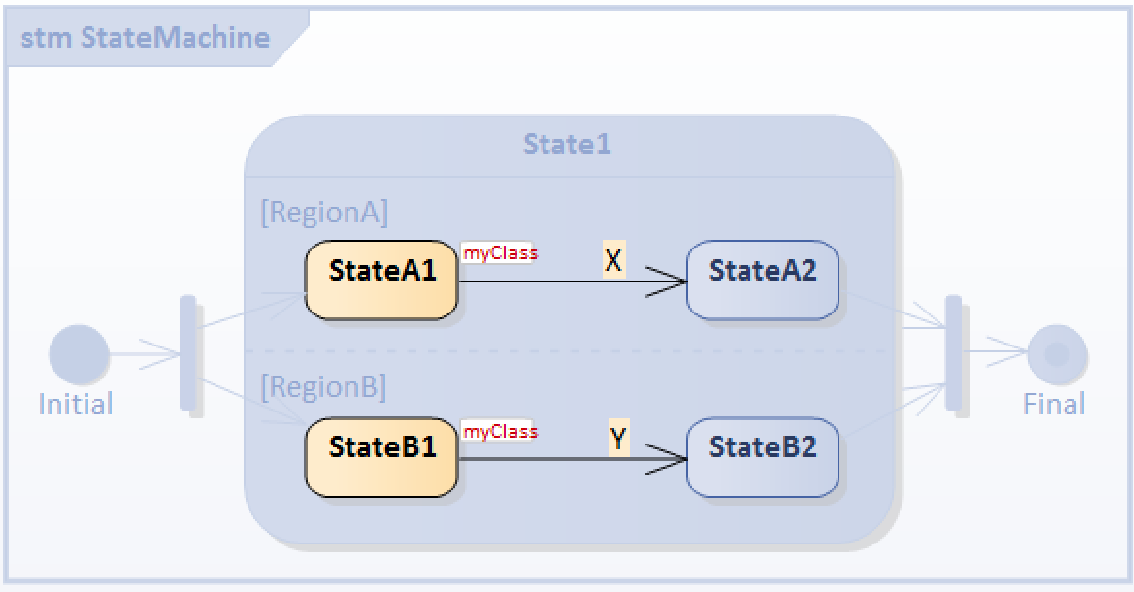
- Cliquez sur *ForkNJoinSimulation* et sélectionnez l'option de ruban 'Simulate > States Exécutables > Statemachine > Générer , Build and Exécuter '
- Spécifiez un répertoire pour le code source généré

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

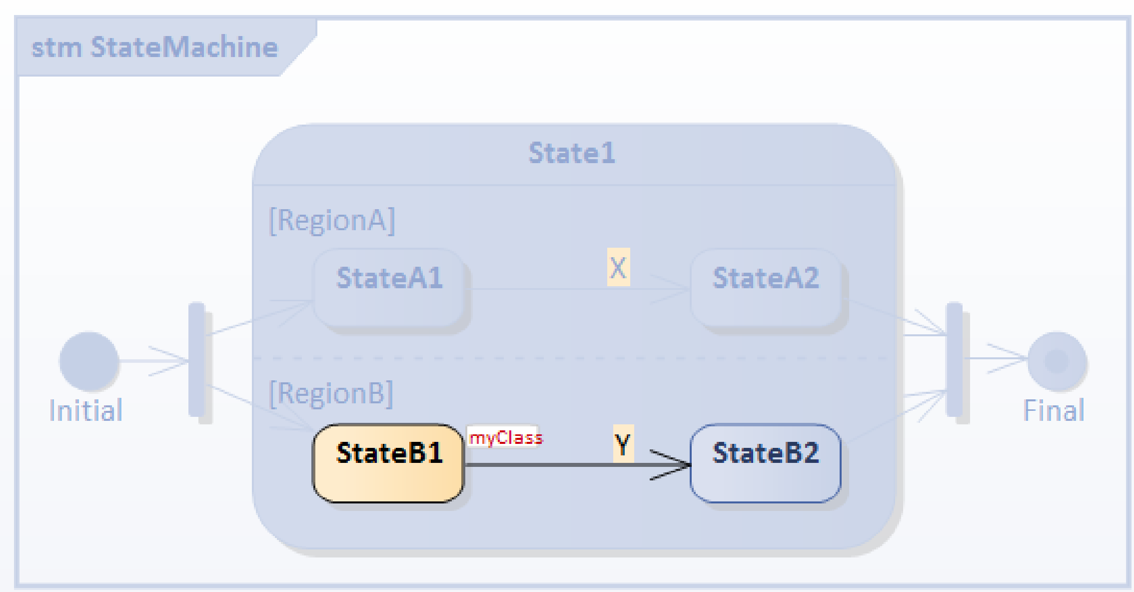
Lorsque la simulation démarre, *State1* , *StateA1* et *StateB1* sont actifs et la Statemachine attend des événements.



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

Lors de l'événement Déclencheur *X* , *StateA1* va sortir et entrer dans *StateA2* ; après que les comportements entry et doActivity ont été exécuter , les événements d'achèvement de *StateA2* sont envoyés et rappelés. Ensuite, la transition de *StateA2* vers le pseudo-état *Join* est activée et parcourue.

Note : *Join* doit attendre que toutes les transitions entrantes soient terminées avant que l'exécution puisse se poursuivre via une transition sortante. Étant donné que la branche de *RegionB* n'est pas terminée (car *StateB1* est toujours actif et en attente de déclencheurs), la transition de *Join* à *Final* ne sera pas exécutée à ce stade. ce moment.



Lors de l'événement Déclencheur *Y* , *StateB1* va sortir et entrer *dans StateB2* ; après que le comportement entry et doActivity a été exécuter , les événements d'achèvement de *StateB2* sont envoyés et rappelés. Ensuite, la transition de *StateB2* vers le pseudo-état *Join* est activée et parcourue. Cela satisfait les critères de tous les transitions entrantes de *Join* étant terminées, 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 du ruban 'Simuler > Simulation Dynamique > Simulateur > Ouvrir la fenêtre Simulation ').

myClass[MyClass].Initial_82285__TO__fork_82286_82286_61745 Effet

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

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. Modifiez le « type » de transition en « interne ».
3. Spécifiez le Déclencheur comme étant l'événement que vous souhaitez différer.
4. Dans le champ « Effet », saisissez « defer(); ».

Pour 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 affiche l'exécution sous forme de texte. Vous pouvez saisir « dump » dans la ligne de commande du simulateur pour afficher le nombre d'événements différés dans la file d'attente. Le résultat peut ressembler à ceci :
   [24850060] Pool d'événements : [NOUVEAU,NOUVEAU,NOUVEAU,NOUVEAU,NOUVEAU,]

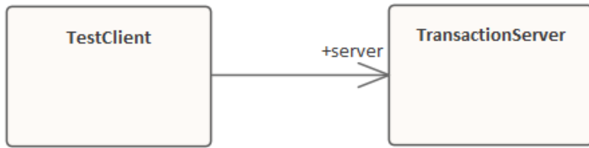
Exemple d'événement différé

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

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

Créer un contexte et Statemachine

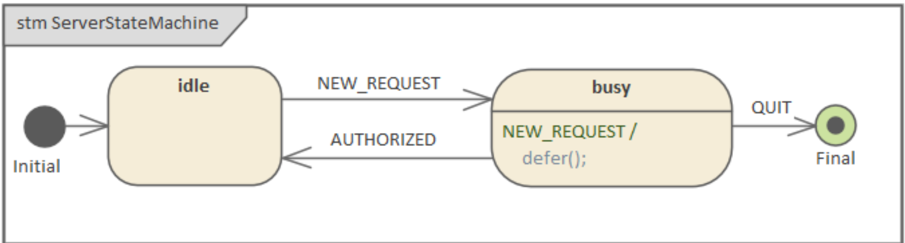
Créer le contexte du serveur



Créez un diagramme de classes et :

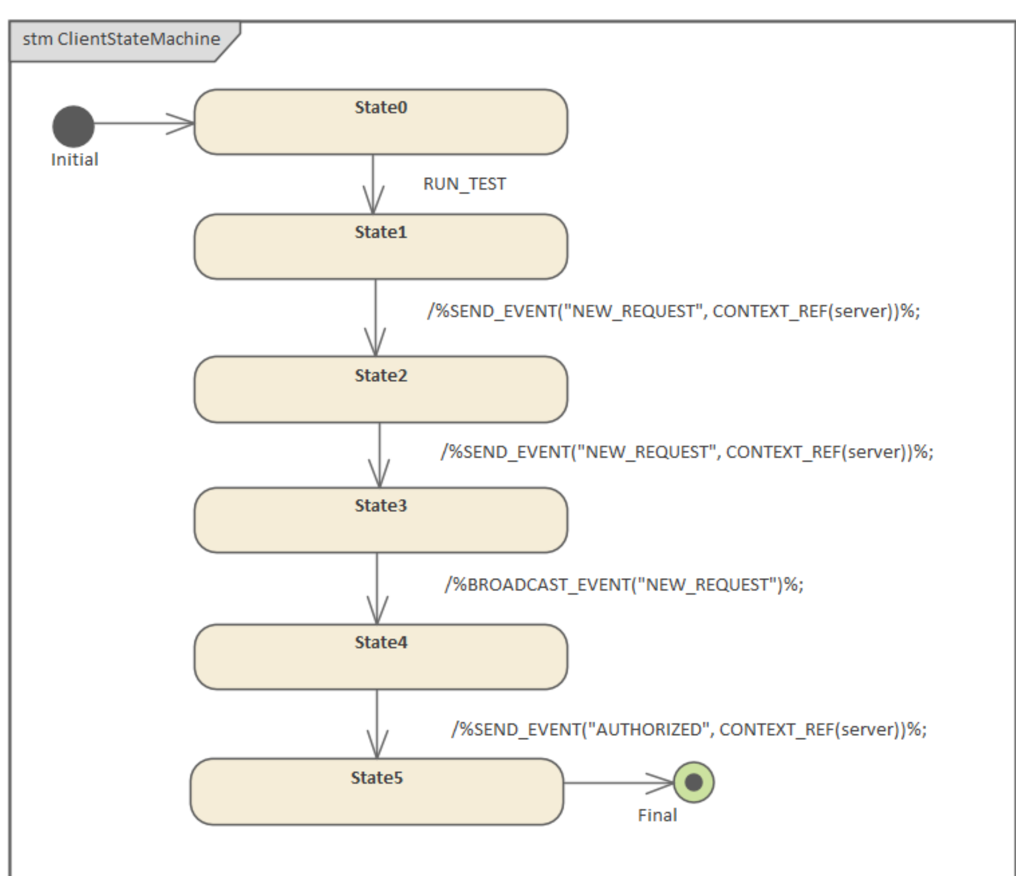
1. Un élément de classe *TransactionServer* , auquel vous ajoutez une Statemachine *ServerStateMachine* .
2. Un élément de classe *TestClient* , auquel vous ajoutez une Statemachine *ClientStateMachine* .
3. Une association de *TestClient* à *TransactionServer* , avec le rôle cible nommé *serveur.*

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 *busy* .
3. Transition (avec événement QUIT comme Déclencheur) vers un State Final Final .
4. Transition (avec événement AUTORISÉ comme Déclencheur) vers *idle* .
5. Transition (avec événement NEW_REQUEST comme Déclencheur et *defer();* comme effet) vers *busy*

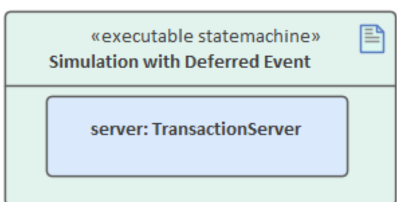
Modélisation pour *ClientStateMachine*



1. Ajoutez un nœud initial *Initial* au diagramme Statemachine et passez à un State *State0* .
2. Transition (avec l'événement RUN_TEST comme déclencheur) vers un State *State1* .
3. Transition (avec effet : %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(serveur))%;) vers un State *State2.*
4. Transition (avec effet : %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(serveur))%;) 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(serveur))%;) vers un State *State5* .
7. Transition vers un State final *final.*

Simulation dans un contexte simple

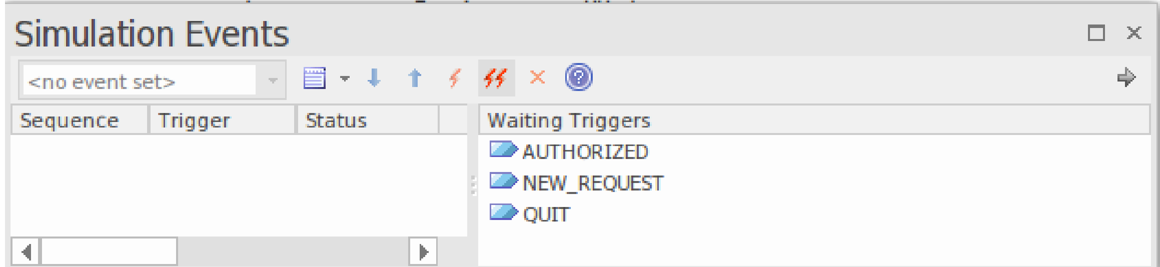
Créer l'artefact Simulation



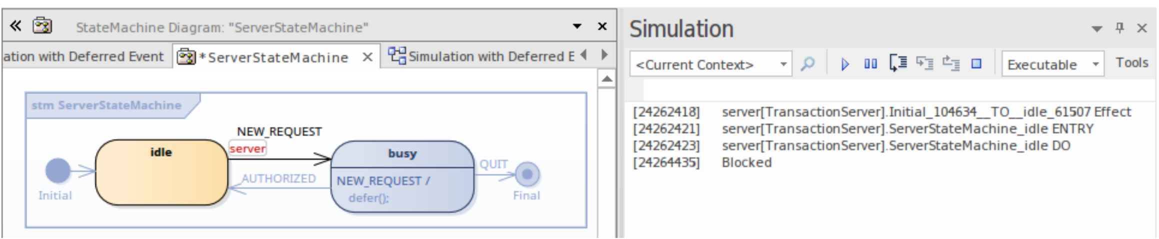
1. Créez un artefact Statemachine Exécutable avec le nom *Simulation avec événement différé* et le champ « Langue » défini sur *JavaScript* .
2. Agrandissez-le, puis appuyez sur Ctrl et faites glisser l'élément *TransactionServer* sur l'artefact et collez-le en tant que propriété avec le *serveur de noms.*

Exécuter la Simulation

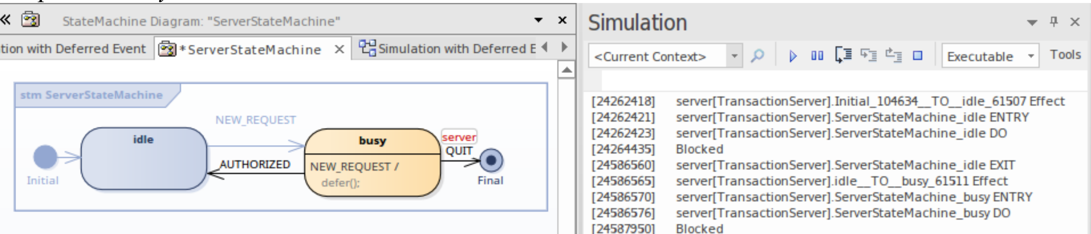
1. Sélectionnez l'Artefact, puis sélectionnez l'option 'Simuler > States Exécutables > Statemachine > Générer , Build et Exécuter ', et spécifiez un répertoire pour votre code (Note : tous les fichiers du répertoire seront supprimés avant le démarrage de la simulation).
2. Cliquez sur le bouton Générer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



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

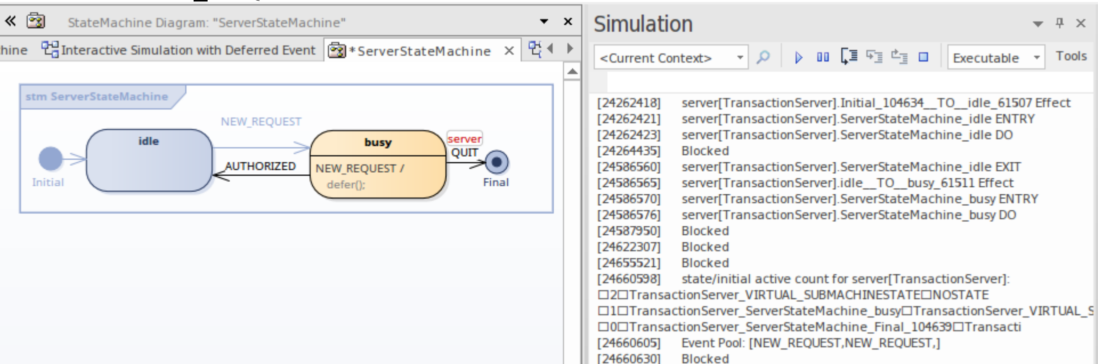


1. Double-cliquez sur NEW_REQUEST dans la fenêtre Simulation Event pour l'exécuter comme le Déclencheur ; *idle* est quitté et *busy* est activé.

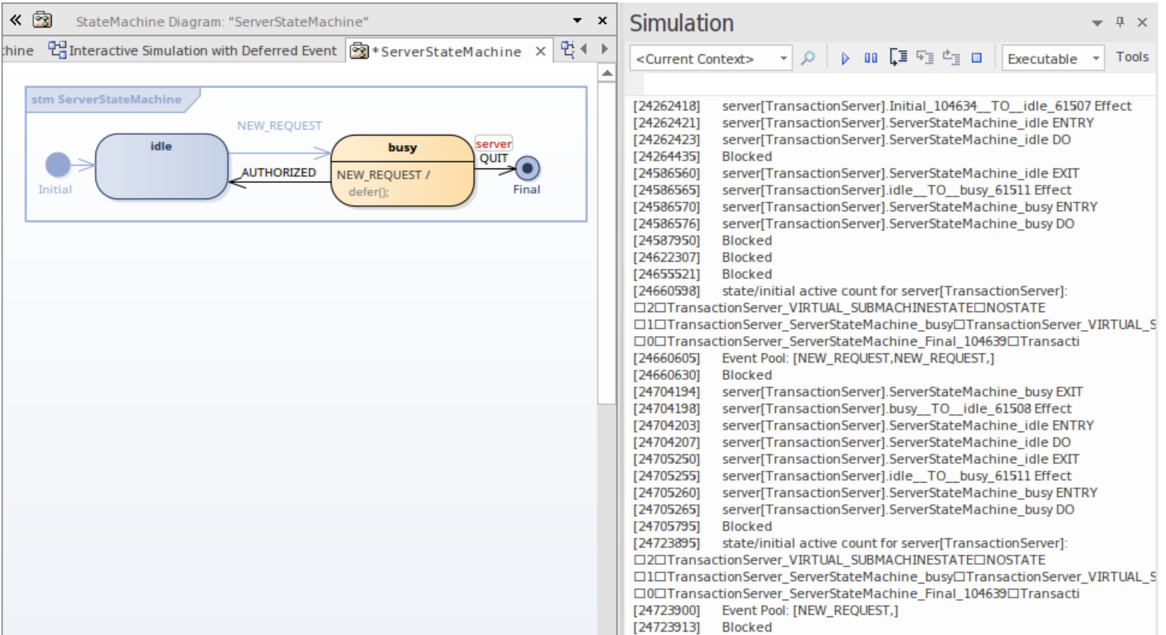


2. Double-cliquez sur NEW_REQUEST dans la fenêtre Événement Simulation pour l'exécuter à nouveau car le Déclencheur ; *busy* reste activé et une instance de NEW_REQUEST est ajoutée dans le pool d'événements.

3. Double-cliquez sur NEW_REQUEST dans la fenêtre Événement Simulation pour l'exécuter une troisième fois pendant que le Déclencheur ; *busy* reste activé, et une instance de NEW_REQUEST est ajoutée dans le pool d'événements.
4. Type *dump* dans la ligne de commande de la fenêtre Simulation ; notez que le pool d'événements comporte deux instances de NEW_REQUEST.

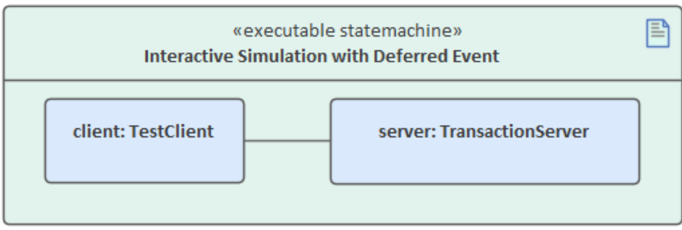


5. Double-cliquez sur AUTORISÉ dans la fenêtre Simulation Event pour l'exécuter en tant que Déclencheur ; ces actions se déroulent :
   - *le mode occupé* est quitté et *le mode inactif* devient actif
   - un événement NEW_REQUEST est récupéré du pool, *l'inactivité* est quittée et *l'occupation* devient active
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 envoi/diffusion d'événement

Créer l'artefact Simulation

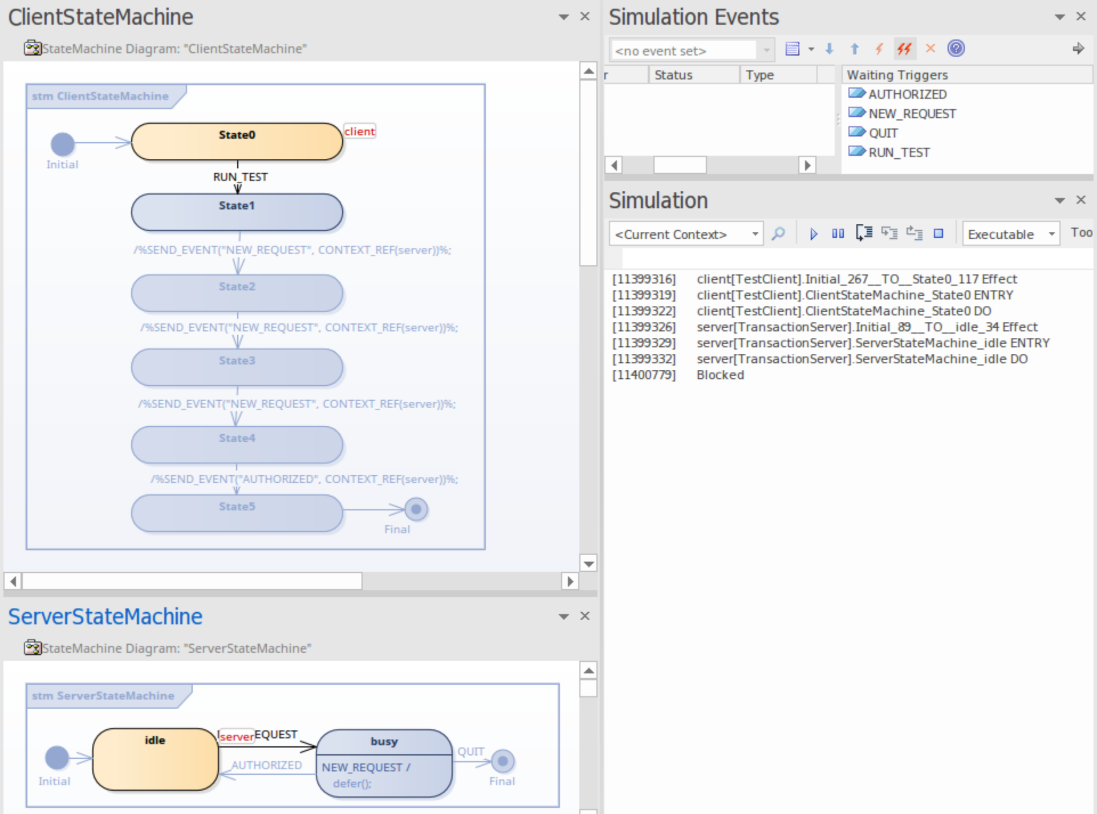


1. Créez un artefact Statemachine Exécutable avec le nom *Simulation interactive avec événement différé* et le champ « Langue » défini sur *JavaScript ;* agrandissez l'élément.
2. Ctrl+faites glisser l'élément *TransactionServer* sur l'artefact et collez-le en tant que propriété avec le *serveur de noms.*
3. Ctrl+Faites glisser l'élément *TestClient* sur l'artefact et collez-le en tant que propriété avec le nom *client* .
4. Créer 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* .

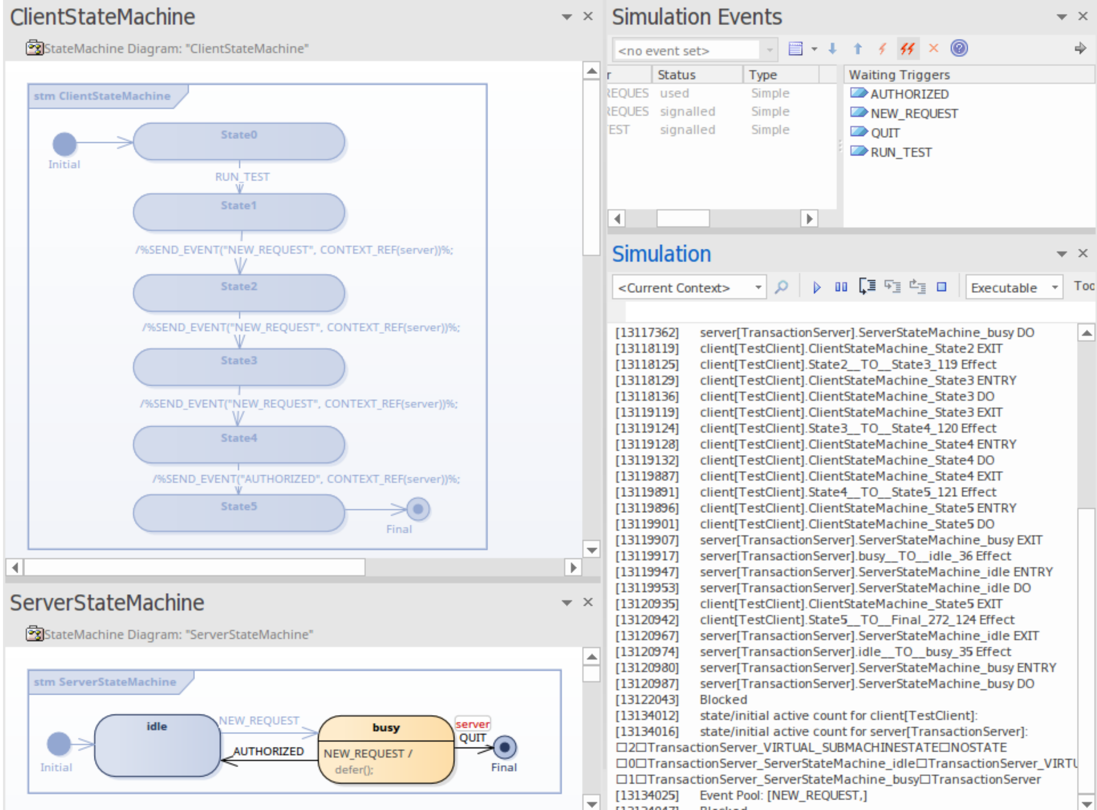
Exécuter Simulation interactive

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

   Une fois la simulation démarrée, le *client* reste à *l'état 0* et le *serveur* reste à *l'état inactif* .



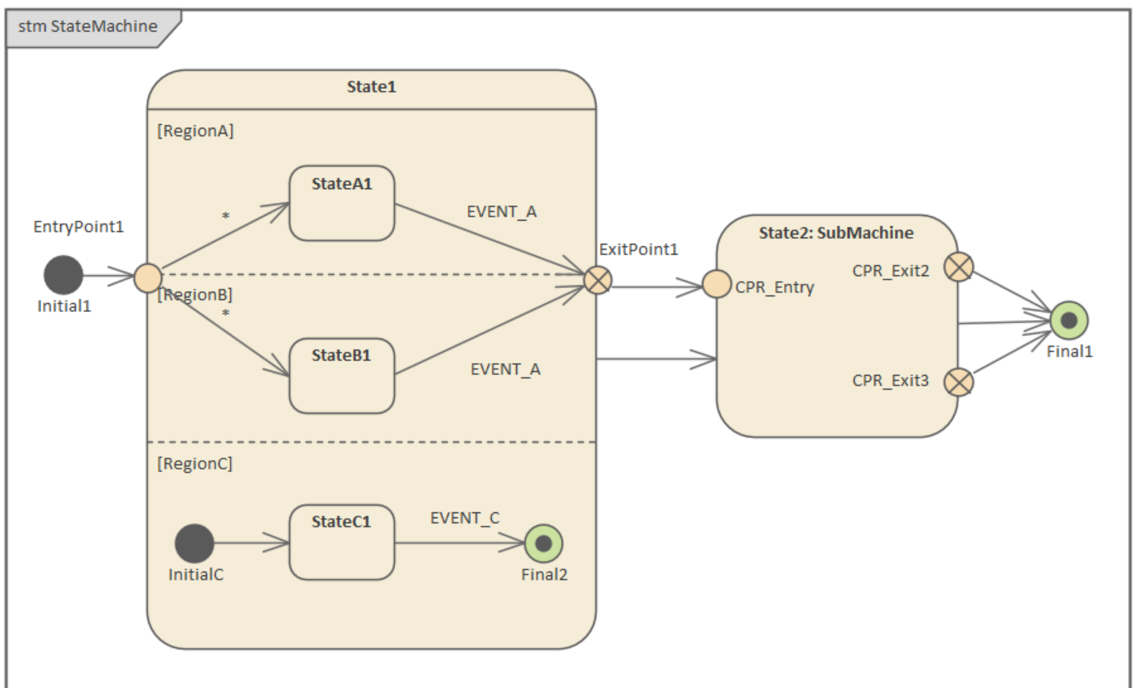
2. Double-cliquez sur RUN_TEST dans la fenêtre Simulation Event pour le déclencheur . L'événement NEW_REQUEST sera déclenché trois fois (par SEND_EVENT et BROADCAST_EVENT) et AUTHORIZED sera déclenché une fois par SEND_EVENT.

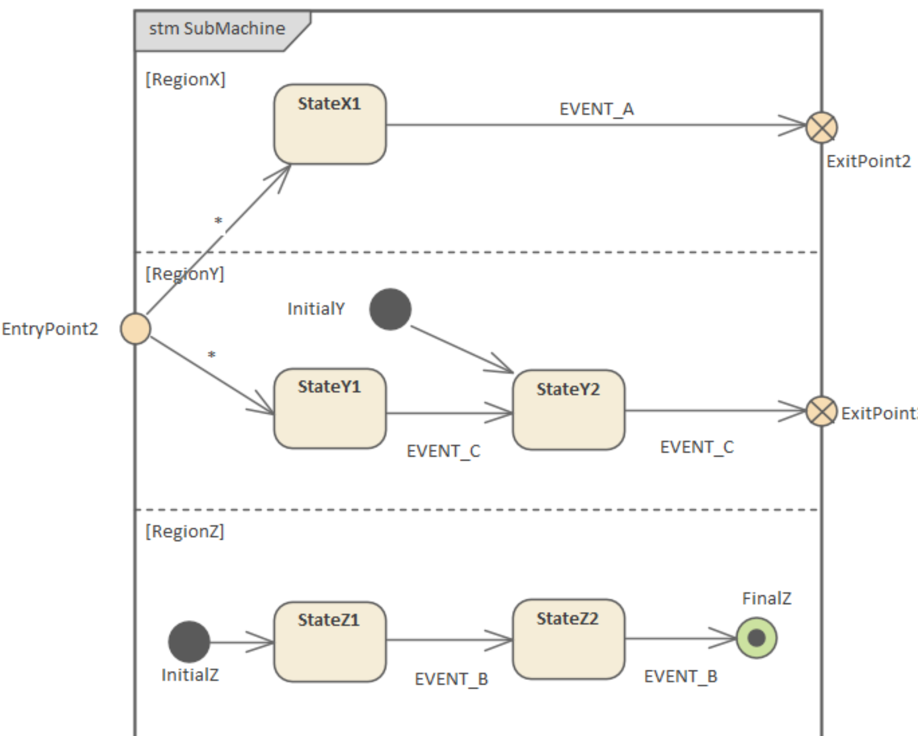


Type *dump* dans la ligne de commande de la fenêtre Simulation . Il reste une instance de NEW_REQUEST dans le pool d'événements. Le résultat correspond à notre test de déclenchement manuel.

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

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





- *L'État 1* est un State composite (également appelé State orthogonal car il comporte plusieurs régions) avec trois

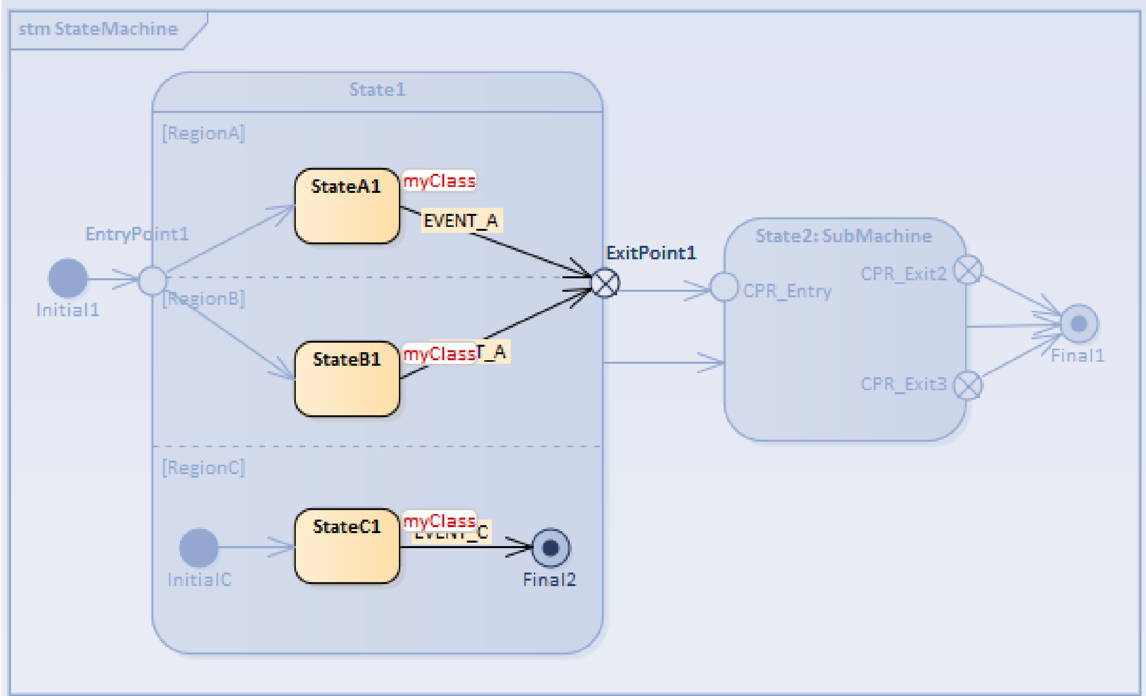
régions : *RégionA* , *RégionB* et *RégionC*

- *State2* est un State de sous-machine appelant *SubMachine* , qui possède 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 : Point d'entrée Entrée

Point d'entrée 1* sur *État 1

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

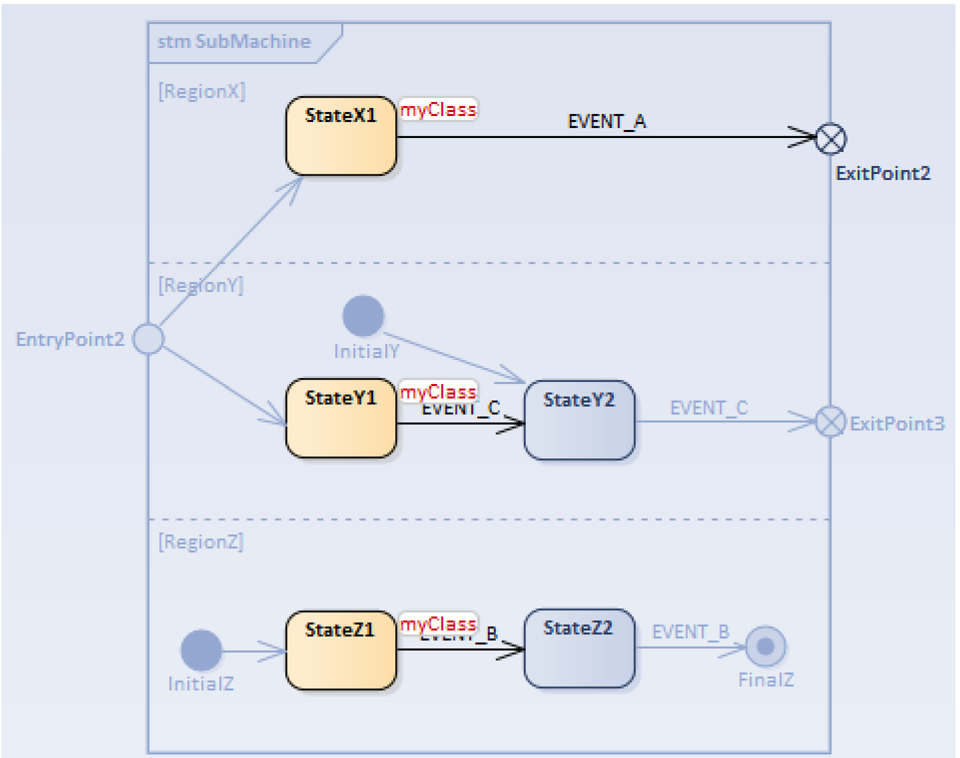


- L'activation explicite se produit pour *RegionA* et *RegionB* , car chacune d'elles est entrée par une transition se terminant sur l'un des sommets contenus dans la région
- L'activation par défaut se produit pour *RegionC* , car elle définit un pseudo-état initial *InitialC* et la transition provenant de l' *InitialC* vers *l'étatC1* démarre l'exécution

EntryPoint2* sur *SubMachine

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

Lorsqu'une transition ciblant 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 chacune d'elles est entrée 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 d' *InitialZ* vers *StateZ1* démarre l'exécution

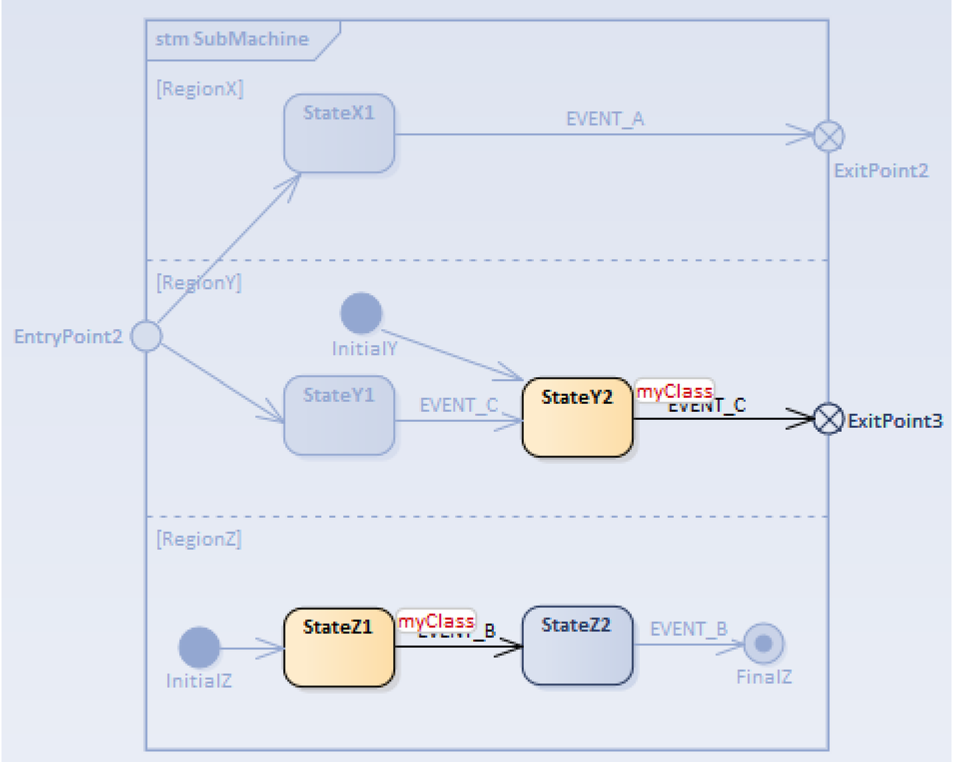
Entrer d'un State : Entrée par défaut

Cette situation se produit 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 ciblant directement *l'État 2* est activée, *l'État 2* est activé, suivi de l'activation par défaut pour toutes les régions de la sous-machine.



- State *de RegionX* est inactif car il ne définit pas de nœud initial
- *La région Y* est activée via *InitialY* et la transition vers *l'état Y2* est exécutée
- *RegionZ* est activé via *InitialZ* et la transition vers *StateZ1* est exécutée

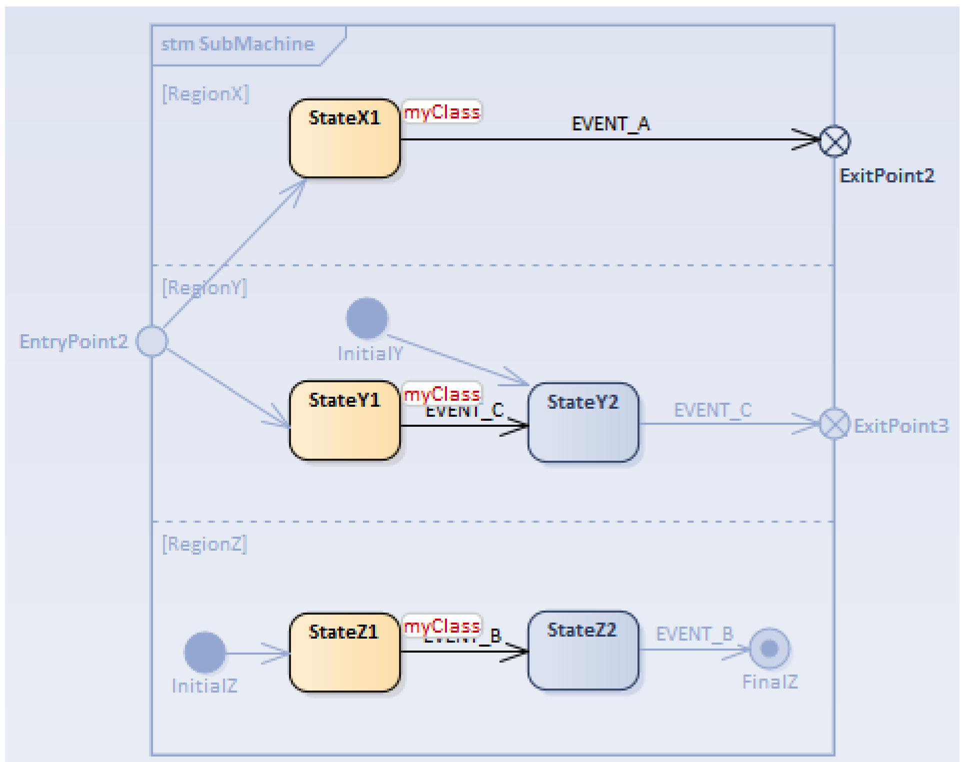
Sortie State

État 1 Sortie

- Séquence Déclencheur [EVENT_C, EVENT_A] : *la région C* est désactivée en premier, puis *la région A* et *la région B* ; une fois le comportement de sortie de *l'état 1* exécuté, la transition sortant de *ExitPoint1* est activée
- Séquence Déclencheur [EVENT_A, EVENT_C] : *les régions A* et *B* sont désactivées en premier, puis *la région C* ; une fois le comportement de sortie de *l'état 1* exécuté, la transition sortant directement de *l'état 1* est activée

Sortie de l'état 2

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



- Séquence Déclencheur [EVENT_A, EVENT_C, EVENT_C, EVENT_B, EVENT_B] : *la région X* est désactivée en premier, puis *la région Y* et *la région Z* en dernier ; une fois le comportement de sortie de *l'état 2* exécuté, la transition sortant directement de *l'état 2* est activée
- Séquence Déclencheur [EVENT_A, EVENT_B, EVENT_B, EVENT_C, EVENT_C] : *RegionX* est désactivé 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*)
- Séquence Déclencheur déclenchement [EVENT_C, EVENT_C, EVENT_B, EVENT_B, EVENT_A] : *la région Y* est désactivée en premier, puis *la région Z* et *la région X* en dernier ; une fois le comportement de sortie de *l'état 2* exécuté, la transition sortant de *CPR_Exit2* est activée (*ExitPoint2* sur *la sous-machine* est lié à *CPR_Exit2* de *l'état 2*)

Exemple : Pseudo-state Historique

L'historique State est un concept pratique associé aux régions d' States composites, selon lequel une région conserve la 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 redevient active (par exemple, après être revenue de la gestion d'une interruption), ou s'il existe une transition locale qui revient à son historique.

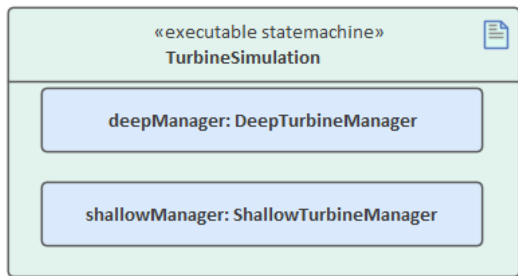
Enterprise Architect supporte deux types de Pseudo-state Historique :

- Historique profond - 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, à la place, terminée sur l' State le plus profond de la configuration de State préservé, y compris l'exécution de tous les comportements d'entrée rencontrés en cours de route
- Historique superficiel - représentant un retour uniquement au sous-état le plus élevé de la configuration State la plus récente, qui est saisi à l'aide de la règle de saisie par défaut

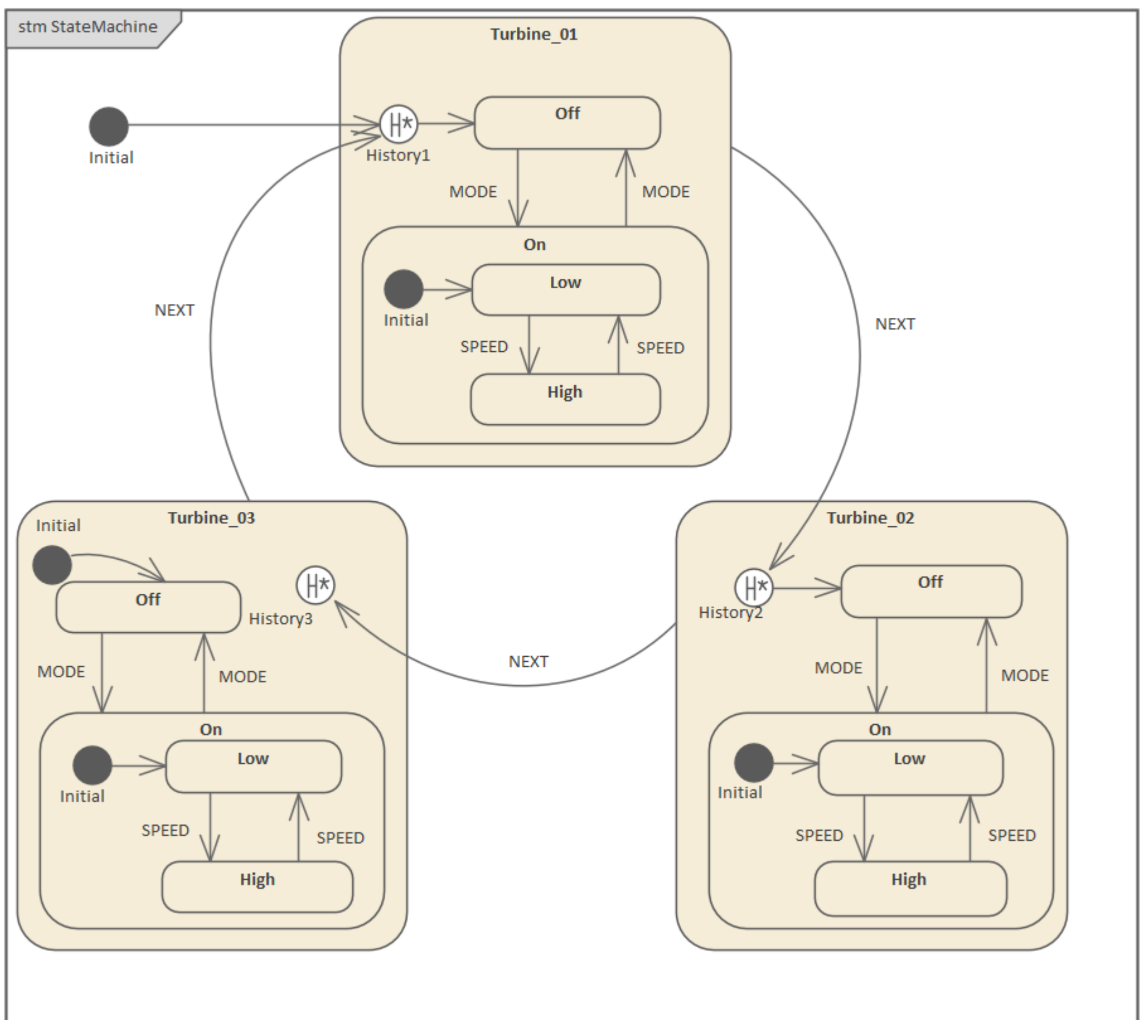
Dans cet exemple, les classes *DeepTurbineManager* et *ShallowTurbineManager* sont exactement les mêmes, sauf que la Statemachine contenue pour la première a un pseudo-état deepHistory et pour la seconde un pseudo-état shallowHistory.

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

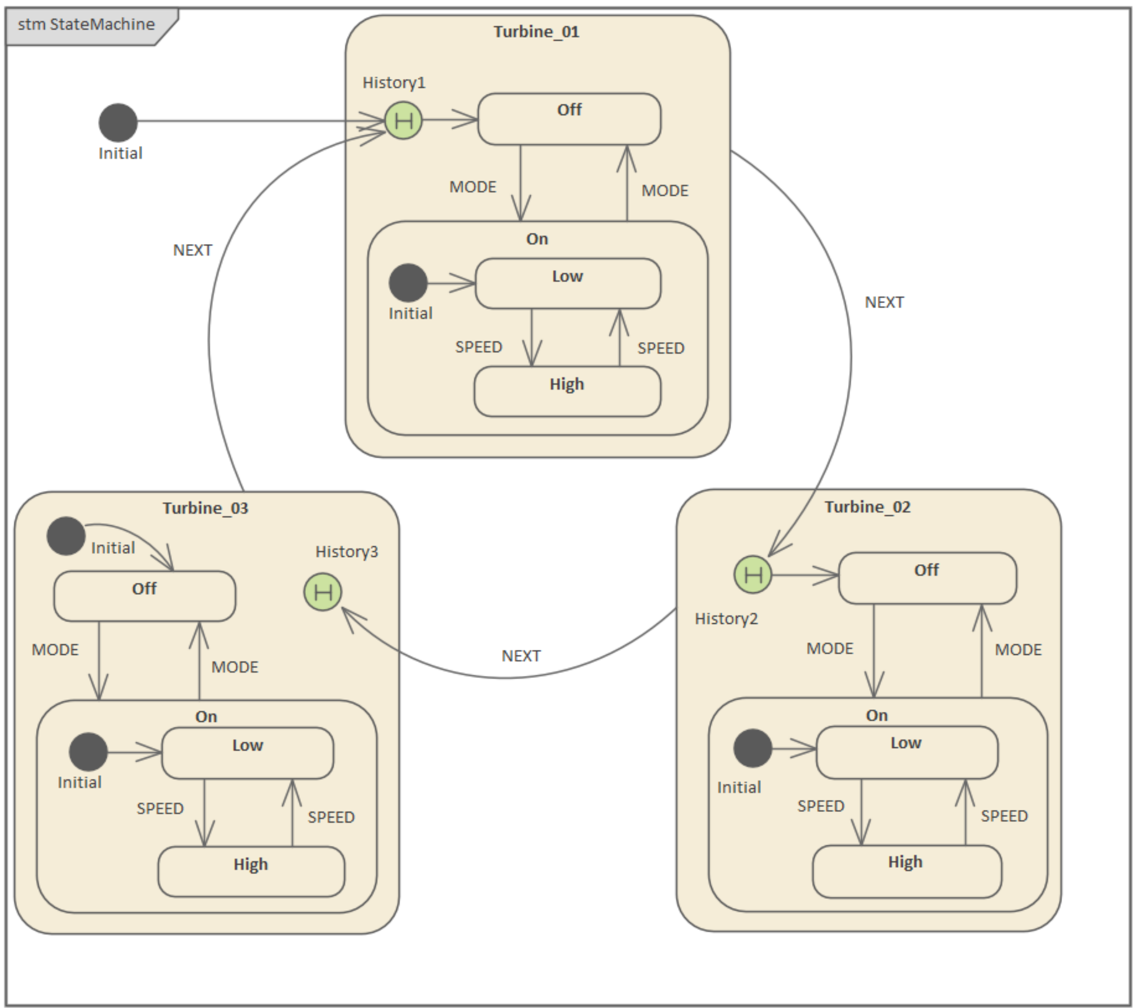
Afin de mieux observer la différence entre l'histoire profonde et l'histoire superficielle, nous exécutons les deux Statemachines dans une seule simulation.



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



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

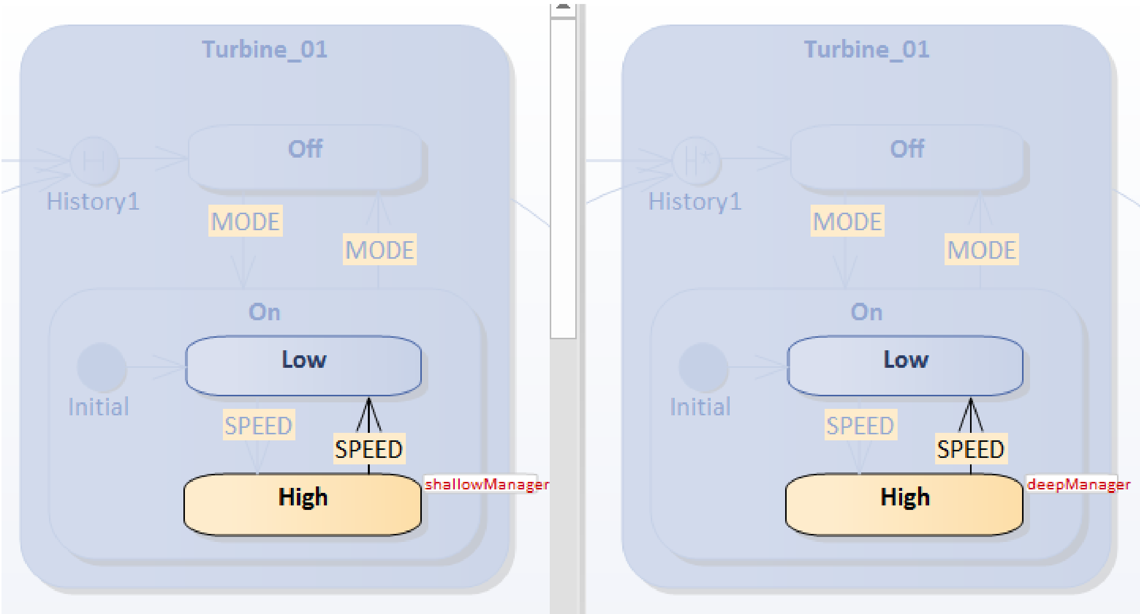


Conseil : Si vous cliquez-droit sur le nœud Historique du diagramme et sélectionnez l'option 'Avancé | Historique profond', vous pouvez basculer le type de Pseudo-state Historique entre superficiel et profond.

Première activation des States

Une fois la simulation démarrée, *Turbine_01* et son sous-état *Off* sont activés.

Séquence Déclencheur : [MODE, SPEED]

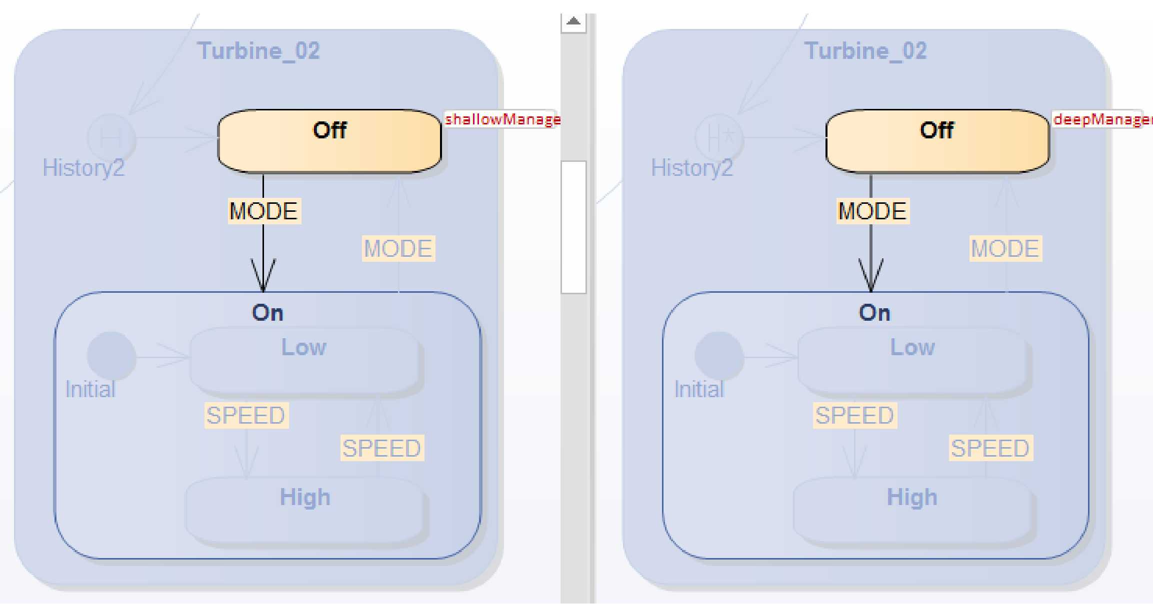


Ensuite, la configuration State actif comprend :

- Turbine_01
- Turbine_01.On
- Turbine_01.En.haut.

Ceci s'applique à la fois *à deepManager* et *à shallowManager.*

Séquence Déclencheur : [SUIVANT]



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

01 peu profondManager [ShallowTurbineManager].StateMachine_Turbine_01_On_High SORTIE

02 peu profondManager [ShallowTurbineManager].StateMachine_Turbine_01_On EXIT

03 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_01 SORTIE

04 effet de ShallowTurbineManager[ShallowTurbineManager].Turbine_01__TO__History2_105720_61730

05 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_02 ENTRÉE

06 gestionnaire peu profond [gestionnaire de turbine peu profond].StateMachine_Turbine_02 DO

07 shallowManager[ShallowTurbineManager].History2_105720__TO__Off_61731 Effet

08 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_02_Off ENTRÉE

09 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_02_Off FAIRE

Note : étant donné que *deepManager* a exactement la même trace que *shallowManager* , la trace de *deepManager* est filtrée à partir de cette séquence.

Nous pouvons apprendre que :

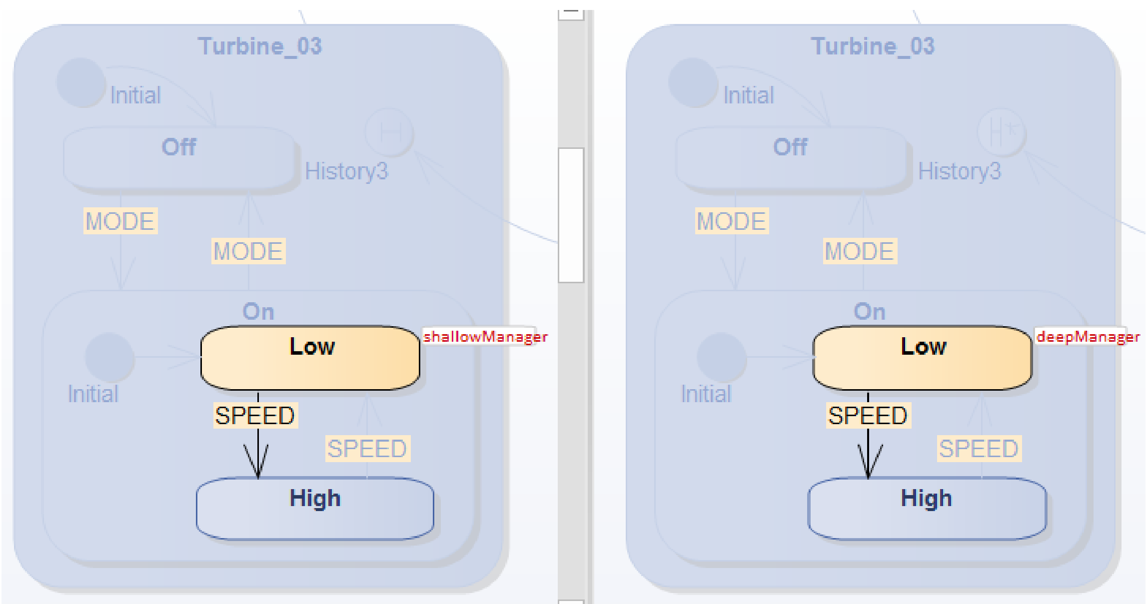
- La sortie d'un State composite commence par l' State le plus interne dans la configuration State actif (voir les lignes 01 à 03 dans la séquence de trace)
- La transition d'historique par défaut n'est prise que si l'exécution mène au nœud Historique (voir ligne 04) et que l' State n'a jamais été actif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

Ensuite, la configuration State actif comprend :

- Turbine_02
- Turbine_02.Arrêt

Ceci s'applique à la fois *à deepManager* et *à shallowManager.*

Séquence Déclencheur : [NEXT, MODE]



Cette séquence de traces peut être observée à partir de la fenêtre Simulation :

Déclencheur [SUIVANT]

01 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off SORTIE

02 Gestionnaire peu profond [Gestionnaire de turbines peu profondes]. StateMachine_Turbine_02 SORTIE

03 effet de ShallowTurbineManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725

04 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRÉE

05 gestionnaire peu profond [gestionnaire de turbine peu profond].StateMachine_Turbine_03 DO

06 shallowManager[ShallowTurbineManager].Initial_105706__TO__Off_61718 Effet

07 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_Off ENTRÉE

08 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_Off FAIRE

Déclencheur [MODE]

Message omis...

Note : étant donné que *deepManager* a exactement la même trace que *shallowManager,* la trace de *deepManager* est filtrée de cette séquence.

Nous pouvons apprendre que :

- Comme il n'y a pas de transition d'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 provenant d' *Initial* est activée (voir ligne 06)

Ensuite, la configuration de l'état actif comprend :

- Turbine_03
- Turbine_03.On
- Turbine_03.Marche.Basse

Ceci s'applique à la fois *à deepManager* et *à shallowManager.*

Entrée dans l'histoire des States

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

Turbine_01

- Turbine_01.On
- Turbine_01.En.haut.

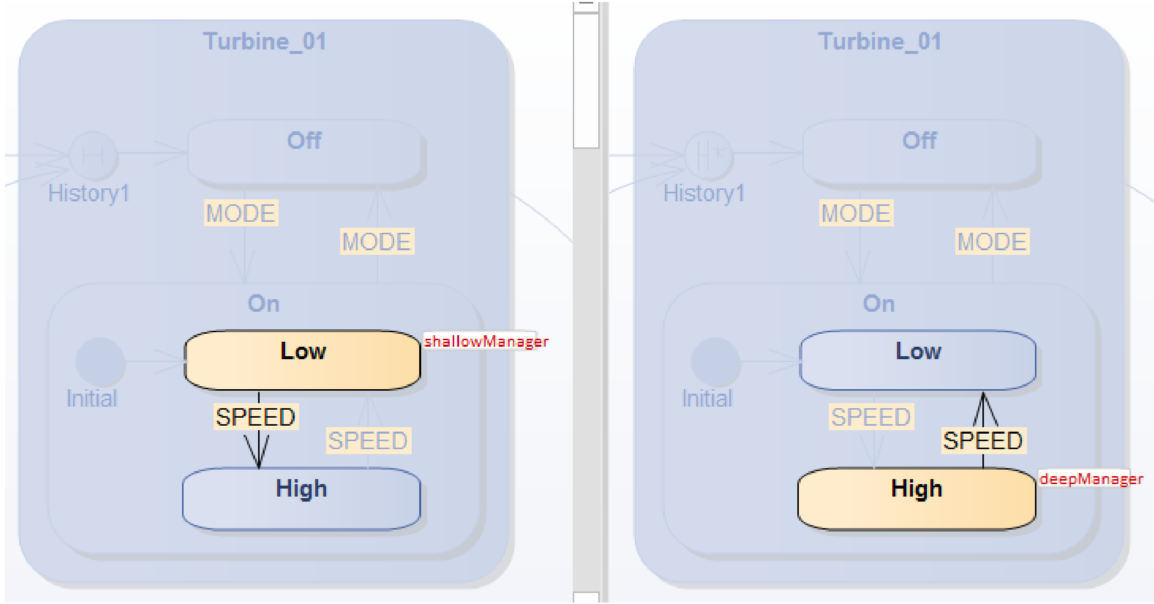
Turbine_02

- Turbine_02.Arrêt

Turbine_03

- Turbine_03.On
- Turbine_03.Marche.Basse

Lorsque nous appuierons sur Déclencheur NEXT, Turbine_01 sera à nouveau activée.



Cette séquence de traces peut être observée à partir de la fenêtre Simulation :

Pour le gestionnaire peu profond :

Déclencheur [SUIVANT]

01 shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low SORTIE

02 peu profondManager [ShallowTurbineManager].StateMachine_Turbine_03_On EXIT

03 Gestionnaire peu profond [Gestionnaire de turbines peu profondes]. StateMachine_Turbine_03 SORTIE

04 effet de ShallowTurbineManager[ShallowTurbineManager].Turbine_03__TO__History1_105711_61732

05 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_01 ENTRÉE

06 gestionnaire peu profond [gestionnaire de turbine peu profond].StateMachine_Turbine_01 DO

07 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_01_On ENTRÉE

08 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_01_On FAIRE

09 effet de ShallowTurbineManager[ShallowTurbineManager].Initial_105721__TO__Low_61729

10 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low ENTRÉE

11 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low DO

Nous pouvons 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 s'est activé à *Low*

Pour deepManager :

Déclencheur [SUIVANT]

01 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low SORTIE

02 deepManager[DeepTurbineManager].StateMachine_Turbine_03_À la sortie

03 deepManager[DeepTurbineManager].StateMachine_Turbine_03 SORTIE

04 deepManager[DeepTurbineManager].Turbine_03__TO__History1_105679_61708 Effet

05 deepManager[DeepTurbineManager].StateMachine_Turbine_01 ENTRÉE

06 deepManager[DeepTurbineManager].StateMachine_Turbine_01 FAIRE

07 deepManager[DeepTurbineManager].StateMachine_Turbine_01_On ENTRÉE

08 deepManager[DeepTurbineManager].StateMachine_Turbine_01_On_High ENTRÉE

Nous pouvons apprendre que :

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

Déclencheur [NEXT] pour sortir de Turbine_01 et activer Turbine_02

Les deux *shallowManager* et *deepManager* activent Turbine_02.Off, qui est l'historique instantané lorsqu'ils sont sortis.

Déclencheur [NEXT] pour sortir de la Turbine_02 et activer la Turbine_03

Les *fonctions shallowManager* et *deepManager* activent toutes deux Turbine_03.On.Low. Cependant, les séquences de *ces fonctions* sont différentes.

Pour *le gestionnaire de profondeur* , le *paramètre shallowHistory* ne peut restaurer que jusqu'à Turbine_03.On. Étant donné qu'un nœud *initial* est défini dans Turbine_03.On, la transition provenant d' *Initial* sera activée et Turbine_03.On.Low sera atteinte.

01 ShallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off SORTIE

02 Gestionnaire peu profond [Gestionnaire de turbines peu profondes]. StateMachine_Turbine_02 SORTIE

03 effet de ShallowTurbineManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725

04 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRÉE

05 gestionnaire peu profond [gestionnaire de turbine peu profond].StateMachine_Turbine_03 DO

06 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_On ENTRÉE

07 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_On FAIRE

08 effet de ShallowTurbineManager[ShallowTurbineManager].Initial_105727__TO__Low_61728

09 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low ENTRÉE

10 peu profondManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low DO

Pour *deepManager* , le *deephistory* peut restaurer jusqu'à Turbine_03.On.Low directement.

01 deepManager[DeepTurbineManager].StateMachine_Turbine_02_Off SORTIE

02 deepManager[DeepTurbineManager].StateMachine_Turbine_02 SORTIE

03 deepManager[DeepTurbineManager].Turbine_02__TO__History3_105680_61701 Effet

04 deepManager[DeepTurbineManager].StateMachine_Turbine_03 ENTRÉE

05 deepManager[DeepTurbineManager].StateMachine_Turbine_03 FAIRE

06 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On ENTRÉE

07 deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low ENTRÉE

Macro d'événement : EVENT_PARAMETER

EVENT_PARAMETER est une macro de fonction utilisée pour accéder aux attributs d'une instance de signal, dans le comportement de State , la protection et l'effet de la transition. Cette macro sera étendue au code exécutable selon le langage de simulation.

Accès 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(Type de signal, Nom d'attribut du signal)%

Par exemple : un signal « MySignal » possède 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))%

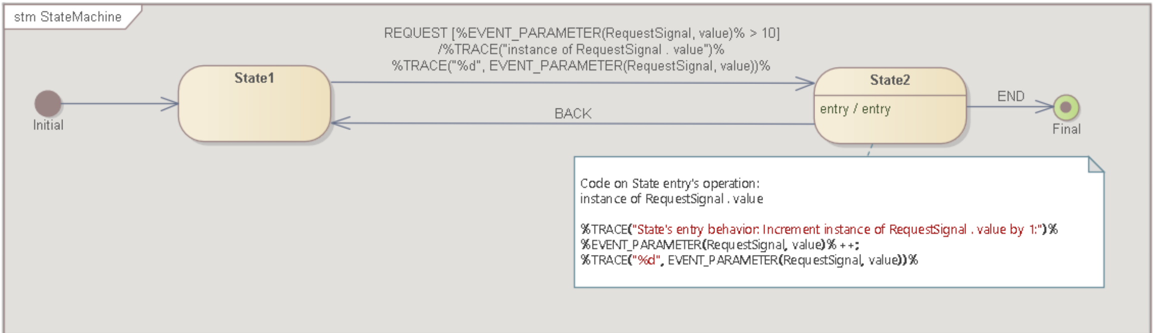
Exemple d'extension macro

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

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

Exemple

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



Pendant l'exécution de la simulation,

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

Étant donné que la condition de garde est fausse, l’état actif restera State1.

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

Étant donné que la condition de garde est vraie, l’état actif passera de l’État 1 à l’État 2 ;

L'effet de 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 de l'attribut du signal et l'avons tracée jusqu'à la fenêtre de simulation.

[32608107] [Partie 1 : TransactionServer] Effet de transition : Initial_4019__TO__State1_4420

[32608118] [Partie 1 : TransactionServer] Comportement de l'entrée : StateMachine_State1

[32608124] [Partie 1 : TransactionServer] Comportement : 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 envoyé : 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 envoyé : REQUEST.RequestSignal(valeur :11)

[32619557] [Partie 1 : TransactionServer] Comportement de sortie : StateMachine_State1

[32619562] [Partie 1 : TransactionServer] Effet de transition : State1__TO__State2_4421

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

[32619571] **11**

[32619576] [Partie 1 : TransactionServer] Comportement de l'entrée : StateMachine_State2

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

[32619590] **12**

[32619594] [Partie 1 : TransactionServer] Comportement : StateMachine_State2

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

[32620211] En attente du Déclencheur

[32622266] Commande : diffusion END

[32622272] [Partie 1 : TransactionServer] Événement mis en file d'attente : FIN

[32622310] [Partie 1 : TransactionServer] Événement envoyé : END

[32622349] [Partie 1 : TransactionServer] Comportement de sortie : StateMachine_State2

[32622359] [Partie 1 : TransactionServer] Effet de transition : State2__TO__Final_4023_4423

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

Limitations et solutions de contournement

Étant donné que l'extension de macro implique un transtypage, il est de la responsabilité de l'utilisateur de s'assurer que le transtypage est valide.

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

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

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

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

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

Par exemple, TransitionA et TransitionB ciblent toutes 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 pour les deux cas.

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

- **Personnaliser gabarit et le code généré**

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