



Lycée La Fayette

Champagne-sur-Seine • Fontaineroux

Département SN-IR



Qt Creator Modeling

© Alain Menu – édition 1.0, avril 2020



Objectifs : Rappels de cours sur UML,

Utiliser Qt Creator pour générer la documentation UML d'un projet...

Cette création est mise à disposition selon le Contrat *Attribution-NonCommercial-ShareAlike 2.0* France disponible en ligne <http://creativecommons.org/licenses/by-nc-sa/2.0/fr/> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.



Table des matières

Introduction.....	3
UML : quelques rappels.....	3
Qt Creator et UML.....	4
Création de modèles.....	5
Ajout d'éléments sur le modèle.....	6
Ajout de relations entre éléments.....	6
Utilisation de l'éditeur de modèles.....	7
Regroupement d'éléments.....	7
Alignement d'éléments.....	7
Suppression d'éléments.....	7
Échelle d'affichage des diagrammes.....	7
Impression des diagrammes.....	8
Exporter des diagrammes en tant qu'images.....	8
Sélection d'un diagramme.....	8
Création des diagrammes d'expression du besoin.....	8
Diagramme de paquetages.....	9
Diagramme de cas d'utilisation.....	9
Description textuelle des cas d'utilisation.....	10
Les diagrammes d'architecture.....	11
Diagrammes de déploiement.....	11
Les diagrammes de classes.....	13
Représentation des classes.....	13
Détails des membres.....	13
Relations entre classes.....	14
Les diagrammes de séquences.....	16
Représentation des objets.....	16
Les messages.....	16
Les fragments d'interaction.....	17
Personnalisation de l'éditeur de modèles.....	19
Diagrammes d'expression du besoin : acteur non humain.....	19
Diagramme de package : relation de communication avec les packages.....	20
Diagramme de cas d'utilisation : relations d'héritage.....	20
Nouvel item de type « note ».....	20
Modification d'aspect des éléments Node et NodeInstance.....	21
Messages de retour des diagrammes de séquences.....	22
Fragments d'interaction.....	22

Références

Site officiel Qt :	https://doc.qt.io/qtcreator/creator-modeling.html
Site officiel UML :	https://www.uml-diagrams.org
OpenClassrooms :	https://openclassrooms.com/fr/courses/2035826
Blog createely:	https://createely.com/blog/diagrams/uml-diagram-types-examples/
Cours en ligne :	http://remy-manu.no-ip.biz/



UML : quelques rappels...

Comme n'importe quel type de projet, un projet informatique nécessite une phase d'analyse, suivi d'une étape de conception.

Dans la phase d'analyse, on cherche d'abord à bien comprendre et à décrire de façon précise les besoins des utilisateurs ou des clients. Que souhaitent-ils faire avec le logiciel ? Quelles fonctionnalités veulent-ils ? Pour quel usage ? Comment l'action devrait-elle fonctionner ? C'est ce qu'on appelle « l'analyse des besoins ». Après validation de notre compréhension du besoin, nous imaginons la solution. C'est la partie analyse de la solution.

Dans la phase de conception, on apporte plus de détails à la solution et on cherche à clarifier des aspects techniques, tels que l'installation des différentes parties logicielles à installer sur du matériel.

Pour réaliser ces deux phases dans un projet informatique, nous utilisons des méthodes, des conventions et des notations ; UML (*Unified Modeling Language*) fait partie des notations les plus utilisées aujourd'hui. La notation UML est un langage visuel constitué d'un ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du projet à traiter (il existe au total 14 types différents de diagrammes UML !).

Le langage UML ne préconise aucune démarche, ce n'est donc pas une méthode. Chacun est libre d'utiliser les types de diagramme qu'il souhaite, dans l'ordre qu'il veut. Il suffit que les diagrammes réalisés soient cohérents entre eux...

Expression du besoin

Les besoins des utilisateurs sont décrits à l'aide de deux types de diagramme :

- le **diagramme de paquetages** (*Package Diagram*) qui permet de décomposer le système en catégories ou parties plus facilement observables ;
- les **diagrammes de cas d'utilisation** (*Use Case Diagram*). On peut faire un diagramme de cas d'utilisation pour le logiciel entier ou pour chaque package.

Aspect fonctionnel

L'aspect fonctionnel du logiciel est présenté sous forme d'une **vue logique** et d'une **vue dynamique**.

La vue logique a pour but d'identifier les éléments du domaine, les relations et interactions entre ces éléments. Deux types de diagramme peuvent être utilisés pour cette vue :

- les **diagrammes de classes** (*Class Diagram*) qui, en phase d'analyse, représentent les entités (informations) manipulées par les utilisateurs et qui, en phase de conception, représentent la structure d'un développement orienté objet ;
- les **diagrammes d'objets** (*Object Diagram*) qui servent à illustrer les classes complexes en utilisant des exemples d'instances.

La vue dynamique (ou vue des processus) montre la décomposition du système en processus et actions, les interactions entre les processus et la synchronisation et la communication des activités parallèles. Cette vue peut s'appuyer sur différents diagrammes :

- le **diagramme de séquence** (*Sequence Diagram*) qui permet de décrire les différents scénarios d'utilisation du système ;
- le **diagramme d'activité** (*Activity Diagram*) qui représente le déroulement des actions, sans utiliser les objets. Il est utilisé en phase d'analyse pour consolider les spécifications d'un cas d'utilisation ;

- le **diagramme de collaboration** (*Communication Diagram*) qui permet de mettre en évidence les échanges de messages entre objets. Il aide si besoin à compléter les diagrammes de séquence et de classes ;
- le **diagramme d'état-transition** (*State-Machine Diagram*) qui sert à décrire le cycle de vie des objets d'une classe.

Architecture du logiciel

L'aspect lié à l'architecture du logiciel permet de définir les composants et les matériels à utiliser. Cet aspect est illustré essentiellement grâce à deux types de diagramme :

- le **diagramme de composants** (*Component Diagram*) décrit tous les composants utiles à l'exécution du système (applications, bibliothèques, instances de base de données, exécutables, etc) ;
- le **diagramme de déploiement** (*Deployment Diagram*) qui correspond à la description de l'environnement d'exécution du système (matériel, réseau...) et de la façon dont les composants y sont installés.

Qt Creator et UML

Qt (prononcez officiellement en anglais *cute* /kju:t/) est un framework orienté objet et développé en C++ par *Qt Development Frameworks*.

L'EDI Qt Creator propose un éditeur de modèle pour créer des modèles de style UML (*Unified Modeling Language*) avec des diagrammes structurés et comportementaux qui fournissent différentes vues de votre système.

Attention : l'éditeur utilise une variante d'UML, seul un sous-ensemble des propriétés du langage est fourni pour spécifier l'apparence des éléments du modèle.

Les diagrammes structurels représentent l'aspect statique du système et sont donc stables, tandis que les diagrammes comportementaux ont des aspects statiques et dynamiques.

Vous pouvez créer les types de **diagrammes structurels** suivants :

- des diagrammes de packages, qui sont constitués de packages et de leurs relations, et visualisent comment le système est conditionné ;
- des diagrammes de classes, qui se composent de classes, de dépendances, d'héritage, d'associations, d'agrégation et de composition, et fournissent une vue orientée objet d'un système ;
- des diagrammes de composants, qui représentent un ensemble de composants et leurs relations, et fournissent une vue d'implémentation d'un système ;
- des diagrammes de déploiement, qui représentent un ensemble de composants logiciels et matériels et leurs relations, et visualisent le déploiement d'un système.

Vous pouvez créer les types de **diagrammes de comportement** suivants :

- des diagrammes de cas d'utilisation, qui se composent d'acteurs, de cas d'utilisation et de leurs relations, et représentent une fonctionnalité particulière d'un système ;
- des diagrammes d'activités, qui visualisent le flux d'une activité à une autre ;
- des diagrammes de séquence, qui se composent d'instances et spécifient où les instances sont activées et détruites et où se termine leur ligne de vie.

Les quatre types de diagramme soulignés dans les listes précédentes sont ceux que nous utilisons en priorité dans le cadre des travaux de BTS SN-IR.

Création de modèles

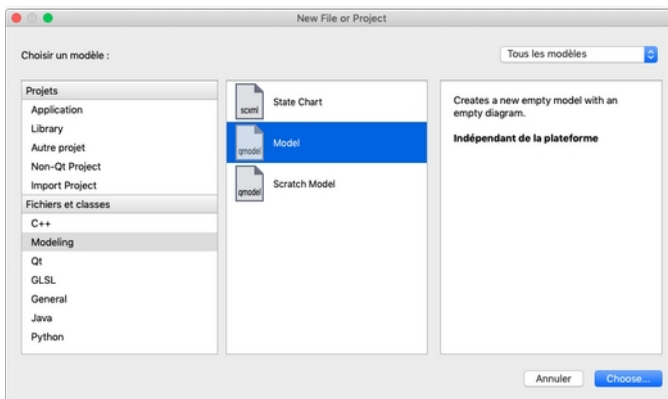
Dans un premier temps, Lancer Qt Creator puis ouvrir ou sélectionner le projet logiciel concerné par la modélisation.

Vous pouvez utiliser des assistants pour créer des modèles et des modèles de travail (*scratch models*). Un *scratch model* peut-être utilisé pour assembler rapidement un diagramme temporaire.

Pour créer des modèles :

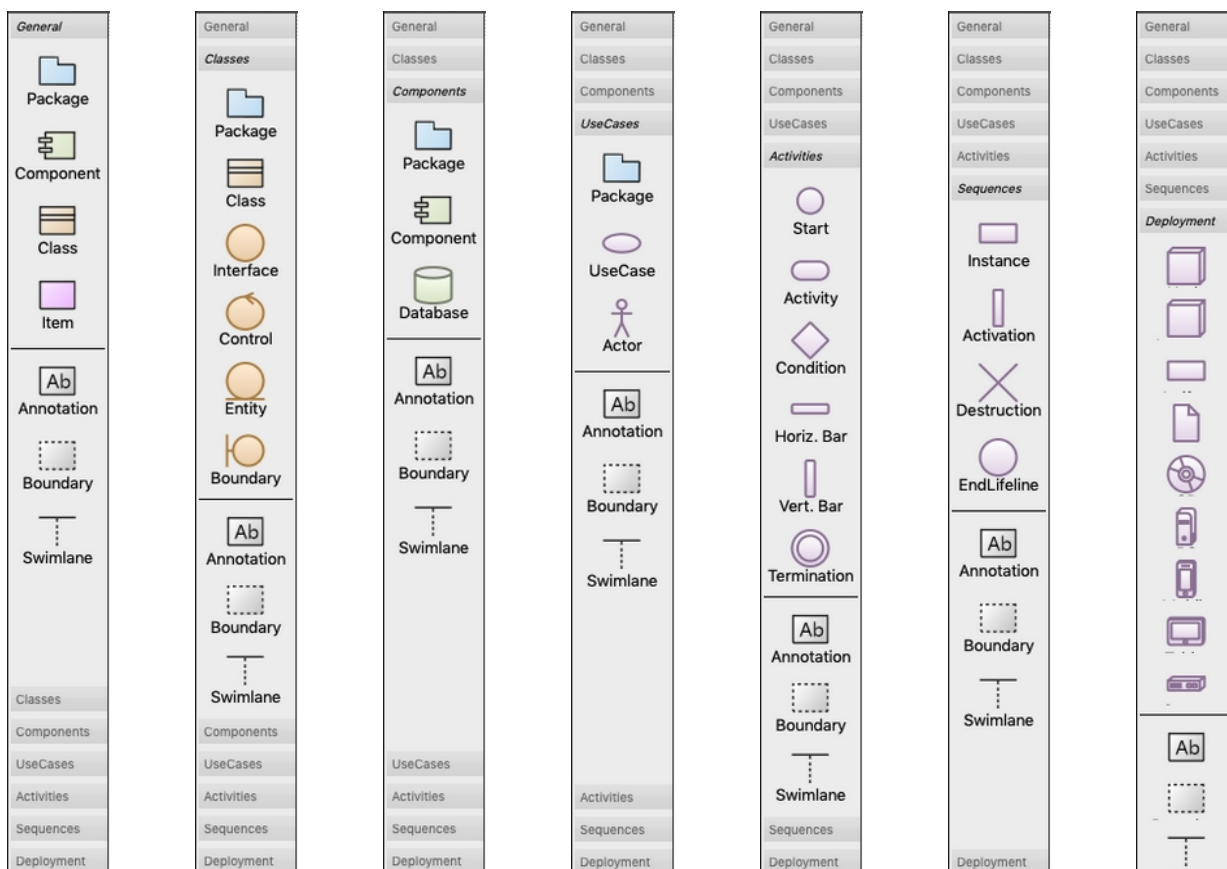
Sélectionnez **Fichier > Nouveau fichier ou projet... > Modélisation** (*File > New File or Project... > Modeling*).

Choisissez Modèle (*Model*) pour un diagramme intégré au projet logiciel ou Modèle de travail (*Scratch Model*) pour un diagramme indépendant (dans ce dernier cas, il conviendra de sauvegarder le fichier à l'endroit désiré).



Choisissez ensuite un nom pour le modèle (extension automatique *.qmodel*) et une localisation. Par défaut, le fichier modèle est ajouté dans la liste DISTFILES du fichier projet *.pro* courant ; il est localisé dans le répertoire des sources du projet.

La zone d'édition du diagramme comporte une palette à onglets à gauche, une vue arborescente et une fenêtre de propriétés à droite. Les onglets de la palette se présentent comme suit :



Ajout d'éléments sur le modèle

Faites glisser et déposez les éléments du modèle dans l'éditeur et sélectionnez-les pour spécifier leurs propriétés :

Dans le champ **Stéréotypes** (*Stereotypes*), entrez le stéréotype à appliquer à l'élément de modèle ou sélectionnez un stéréotype prédéfini dans la liste.

Dans le champ **Nom** (*Name*), donnez un nom à l'élément de modèle.

Dans le champ d'**Affichage du stéréotype** (*Stereotype Display*), sélectionnez :

- **Intelligent** (*Smart*) pour afficher le stéréotype sous forme d'étiquette, de décoration ou d'icône, selon les propriétés de l'élément. Par exemple, si une classe a le stéréotype interface, elle est affichée sous forme d'icône jusqu'à ce qu'elle devienne un membre affiché, après quoi elle est affichée comme décoration ;
- **Aucun** (*None*) pour supprimer l'affichage du stéréotype ;
- **Libellé** (*Label*) pour afficher le stéréotype sous forme de ligne de texte en utilisant le formulaire standard au-dessus du nom de l'élément même si le stéréotype définit une icône personnalisée ;
- **Décoration** (*Decoration*) pour montrer la forme standard de l'élément avec le stéréotype comme une petite icône placée en haut à droite si le stéréotype définit une icône personnalisée ;
- **Icône** (*Icon*) pour afficher l'élément à l'aide de l'icône personnalisée.

Les autres propriétés concernent l'aspect visuel de l'élément. Dans le champ **Rôle** (*Role*), sélectionnez un rôle pour rendre la couleur de l'élément de modèle plus claire, plus sombre ou plus douce. Vous pouvez également supprimer la couleur et dessiner le contour de l'élément ou aplatir l'élément en supprimant les dégradés.

Ajout de relations entre éléments

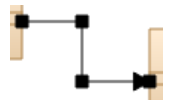
Pour créer une relation entre deux éléments, sélectionnez l'élément de début de la relation puis une icône de flèche située à côté de l'élément et faites-la glisser jusqu'au point de fin de la relation.



Pour déplacer la fin d'une relation vers un autre élément, sélectionnez la relation puis saisissez le point de fin et déposez-le sur un autre élément qui accepte les relations de ce type. Par exemple, seules les classes acceptent les héritages et les associations.

Pour créer des points de brisure (*sampling points*) qui divisent une relation en deux lignes connectées, sélectionnez une relation et appuyez sur **Maj** (*Shift*) et cliquez sur la ligne de relation.

Si possible, le point final d'une relation est déplacé automatiquement pour tracer la ligne jusqu'au prochain point de brisure verticalement ou horizontalement. Pour supprimer un point de brisure, appuyez sur **Ctrl** et cliquez sur le point.



Sélectionnez la relation pour lui spécifier des paramètres selon son type : héritage, association, agrégation, composition ou dépendance. Vous pouvez spécifier les paramètres suivants pour les relations de dépendance, qui sont disponibles pour tous les types d'éléments :

- dans le champ **Stéréotypes** (*Stereotypes*), sélectionnez le stéréotype de la relation ;
- dans le champ **Nom** (*Name*), donnez un nom à la relation ;
- dans le champ **Direction** (*Navigable*), vous pouvez changer la direction de la connexion ou la rendre bidirectionnelle.

Regroupement d'éléments

Pour regrouper des éléments, faites glisser un élément **Frontière** (*Boundary*) vers l'éditeur et redimensionnez-le pour enfermer les éléments dans le groupe.

L'ensemble peut ensuite être déplacé par une simple sélection (sans qu'il soit nécessaire de sélectionner tous les éléments internes).



L'utilisation des couloirs (*swimlanes*) peut aussi grandement faciliter le déplacement de groupes d'éléments graphiques.

Un couloir vertical est créé lorsque vous déposez l'icône de couloir sur la bordure supérieure du diagramme et un couloir horizontal est créé lorsque vous déposez l'icône près de la bordure gauche.



Le couloir apparaît sous forme d'une ligne pointillée. Lorsque vous déplacez le couloir, tous les éléments se trouvant directement à droite du couloir (pour les couloirs verticaux) ou en dessous (pour les couloirs horizontaux) seront déplacés ensemble.

Les classes ou autres objets que vous posez sur les packages sont déplacés avec les packages. Vous pouvez déplacer des éléments individuels et modifier leurs propriétés en les sélectionnant. Vous pouvez également utiliser la sélection multiple pour regrouper temporairement des éléments.

Alignement d'éléments

Pour aligner des éléments dans l'éditeur, sélectionnez plusieurs éléments et cliquez avec le bouton droit pour ouvrir un menu contextuel.

Sélectionnez des actions dans le menu **Aligner les objets** (*Align Objects*) pour aligner les éléments horizontalement ou verticalement ou pour ajuster leur largeur et leur hauteur.

Suppression d'éléments

Pour supprimer un élément, sélectionnez-le puis appuyez sur la touche **Suppr** (*Del*) ou ouvrez le menu contextuel par un clic droit et choisissez **Supprimer** (*Delete*).

Le choix **Enlever** (*Remove*) supprime l'élément de la vue graphique mais le conserve dans la vue arborescente à droite, il pourra donc être re-déposé sur la vue ultérieurement.

Échelle d'affichage des diagrammes



Pour zoomer les diagrammes, sélectionnez le bouton de la barre d'outils **Zoom avant** (*Zoom In*), appuyez sur **Ctrl + +** ou appuyez sur **Ctrl** et faites rouler la molette de la souris vers le haut.

Pour effectuer un zoom arrière sur les diagrammes, sélectionnez **Zoom arrière** (*Zoom Out*), appuyez sur **Ctrl + -** ou appuyez sur **Ctrl** et faites rouler la molette de la souris vers le bas.

Pour réinitialiser la taille du diagramme à 100%, sélectionnez **Réinitialiser le zoom** (*Reset Zoom*) ou appuyez sur **Ctrl + 0**.

Impression des diagrammes

Pour imprimer des diagrammes, appuyez sur **Ctrl + C** lorsqu'aucun élément n'est sélectionné dans l'éditeur pour copier tous les éléments dans le presse-papiers en résolution 300 dpi. Collez ensuite le diagramme dans une application qui peut imprimer des images.

Si vous copiez une sélection d'éléments dans l'éditeur, seuls ces éléments et leurs relations seront copiés dans le presse-papiers en tant qu'image.

Exporter des diagrammes en tant qu'images

Pour enregistrer des diagrammes sous forme d'images, sélectionnez **Fichier > Exporter le diagramme** (*File > Export Diagram*). Pour enregistrer uniquement les parties sélectionnées d'un diagramme, sélectionnez **Exporter les éléments sélectionnés** (*Export Selected Elements*).


Sélection d'un diagramme

Pour ajouter un nouveau diagramme au modèle, cliquez sur .

Les différents diagrammes du modèle apparaissent, dans la vue arborescente en haut à droite, précédés d'une icône orangée.

Double-cliquez sur l'item de l'arbre pour afficher le diagramme.

 MyDiagram

Par défaut, lorsque vous sélectionnez un élément dans un diagramme, il est également mis en surbrillance dans la vue arborescente. Pour que la sélection d'un élément dans la structure le mette également en surbrillance dans le diagramme, cliquez et maintenez le bouton , puis sélectionnez **Synchroniser le diagramme avec la structure** (*Synchronize Diagram with Structure*). Pour conserver les sélections dans le diagramme et la vue structure synchronisées, sélectionnez **Conserver synchronisés** (*Keep Synchronized*).

Création des diagrammes d'expression du besoin

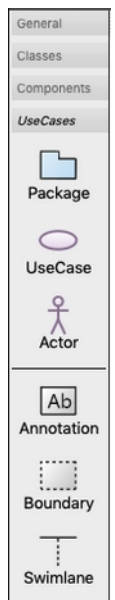
Tout futur logiciel correspond à une boîte noire qui doit fournir des services à son environnement ; donc aux utilisateurs (ou acteurs) de ce logiciel.

Un acteur (humain ou non) correspond à une entité qui aura une interaction avec le système (logiciel). Parmi les acteurs, nous distinguons :

- les acteurs principaux agissent directement sur le système. Il s'agit d'entités qui ont des besoins d'utilisation du système. On peut donc considérer que les futurs utilisateurs du logiciel sont les acteurs principaux ;
- les acteurs secondaires n'ont pas de besoin direct d'utilisation. Ils peuvent être soit consultés par le système à développer, soit récepteur d'informations de la part du système. Cela est généralement un autre système (logiciel) avec lequel le nôtre doit échanger des informations.

Certains acteurs sont de type humain. Ils sont alors représentés par un bonhomme en fil de fer (*stick man*) et on indique leur rôle en dessous.

Pour représenter un acteur de type non-humain, on peut utiliser une représentation graphique différente et/ou ajouter une indication supplémentaire (stéréotype <<system>>).



Note : Qt Model Editor ne permet pas par défaut la mise en place d'acteurs autres que les stick men... se reporter au chapitre « Personnalisation de l'éditeur de modèles ».

Diagramme de paquetages

Les besoins très différents des acteurs et le nombre de fonctionnalités que doit offrir le système étudié peuvent en rendre la vision globale très compliquée. Il est alors souhaitable de décomposer le système en parties distinctes, en fonction des « familles » de fonctionnalités et de façon à pouvoir les analyser séparément. Chacune de ces parties correspond à un domaine fonctionnel nommé **package**.

Un diagramme de paquetages présente les **acteurs** qui interviennent sur les différentes parties. Chaque paquetage est ensuite détaillé par un diagramme de cas d'utilisation.

Vous pouvez ajouter des éléments de package imbriqués à un diagramme de package. La profondeur des éléments du diagramme correspond à la profondeur du modèle structuré. Les éléments empilés sur d'autres éléments du même type sont automatiquement dessinés dans une teinte plus foncée, dans la couleur sélectionnée.

Cliquez avec le bouton droit sur un package pour ouvrir un menu contextuel dans lequel vous pouvez sélectionner **Créer un diagramme** (*Create Diagram*) pour créer un nouveau diagramme de package ou de cas d'utilisation dans le modèle.

Pour mettre à jour les dépendances d'inclusion du package, sélectionnez **Mettre à jour les dépendances d'inclusion** (*Update Include Dependencies*).

Exemple : Boutique en ligne de proximité

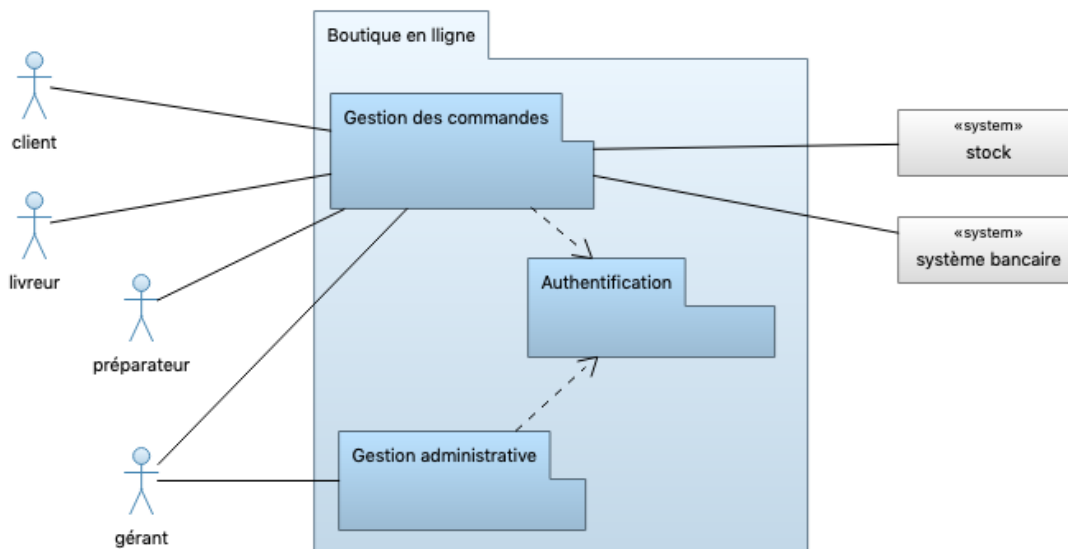


Diagramme de cas d'utilisation

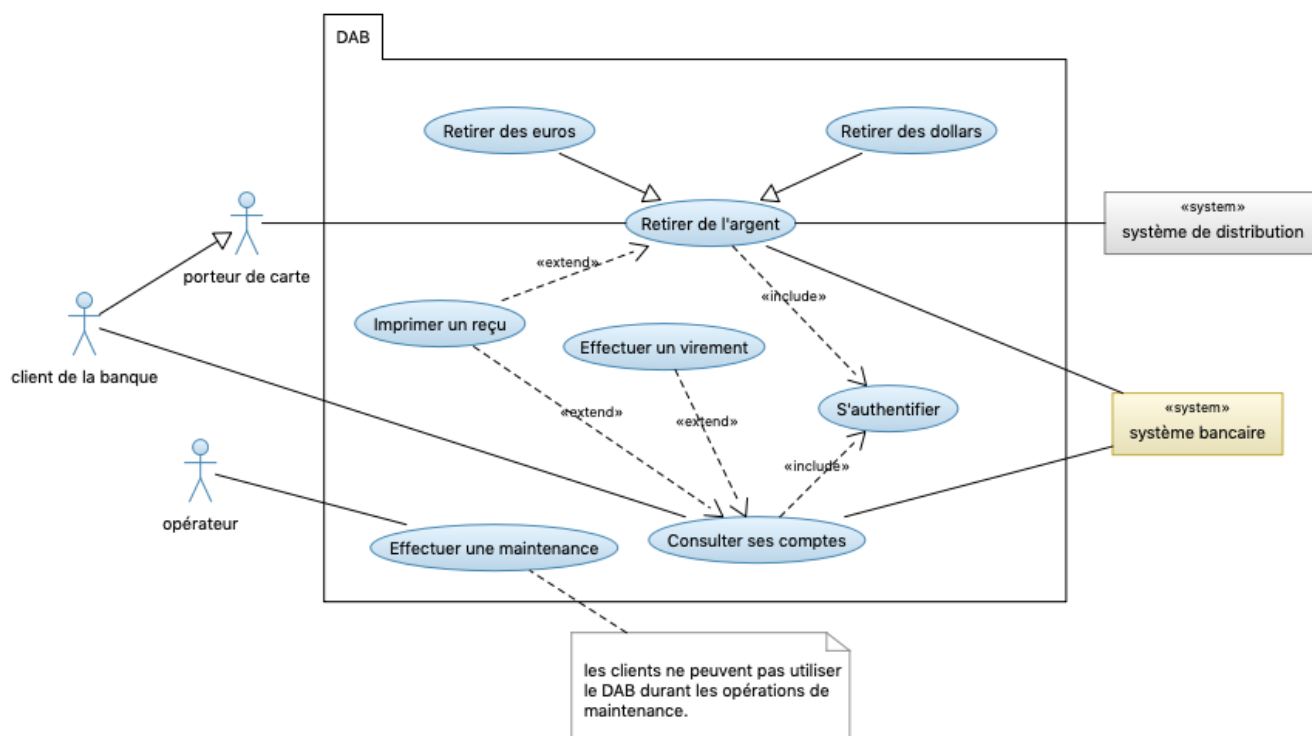
Un diagramme de cas d'utilisation détaille les besoins des acteurs du système étudié ou d'un paquetage de ce système.

Les cas d'utilisation internes sont toujours reliés, par une flèche, au cas d'utilisation initial qui en a besoin. Le cas d'utilisation initial peut être un cas d'utilisation principal (donc directement relié à un acteur) ou à un autre cas d'utilisation interne.

Ces relations sont appelées des **relations stéréotypées**, car elles définissent le type de lien entre les cas d'utilisation, qui peuvent être de deux sortes :

- « *include* » : une relation d'inclusion est utilisée pour indiquer que le cas d'utilisation source (départ de la flèche) contient TOUJOURS le cas d'utilisation inclus ;
- « *extend* » : cette relation est utilisée pour indiquer que le cas d'utilisation source (à l'origine de la flèche) n'est pas toujours nécessaire au cas d'utilisation principal, mais qu'il peut l'être dans certaines situations.

Exemple : Distributeur automatique de billets (zone internationale)



Note : Qt Model Editor ne permet pas par défaut la mise en place de relation d'héritage entre des acteurs ou des UseCases...

La note liée au cas « Effectuer une maintenance » est obtenue par un item personnalisé superposé sur un élément de type Annotation...

... se reporter au chapitre « Personnalisation de l'éditeur de modèles ».

Description textuelle des cas d'utilisation

Pour mémoire, chacun des cas d'utilisation d'un modèle peut être détaillé textuellement lorsque cela est nécessaire à la compréhension du système. Une description textuelle classique comporte trois parties :

1. Identification

Titre : nom du cas d'utilisation, résumé : description du cas d'utilisation, acteurs : descriptions des acteurs principaux et secondaires, dates : date de création et date de mise à jour, responsable : noms du ou des responsables, version : numéro de la version ;

2. Description des scénarios

Pré-conditions : état du système avant que le cas d'utilisation puisse être déclenché, scénarios : un scénario est une instance d'un cas d'utilisation dans lequel tous les paramètres ont été fixés (il y a plusieurs types de scénarios : le **scénario nominal** qui correspond à un déroulement normal d'un cas d'utilisation, les **scénarios alternatifs** qui sont des variantes du scénario normale et les **scénarios d'exceptions** qui décrivent ce qui se passe lors d'une erreur), post-conditions : elles décrivent l'état du système après l'issue de chaque scénario ;

3. Exigences non fonctionnelles

Cette partie est facultative ; mais si elle est présente, elle permet de préciser des spécifications non fonctionnelles (fréquence, fiabilité, type d'interface homme-machine...).

Les diagrammes d'architecture

Les **diagrammes de composants** et les **diagrammes de déploiement** sont deux types de vues statiques en UML. Les premiers décrivent le système modélisé sous forme de composants réutilisables et mettent en évidence leurs relations de dépendance. Les seconds se rapprochent encore plus de la réalité physique, puisqu'ils identifient les éléments matériels (PC, Modem, Station de travail, Serveur, etc.), leur disposition physique (connexions) et la disposition des exécutables (représentés par des composants) sur ces éléments matériels.

Diagrammes de déploiement

Un diagramme de déploiement décrit le déploiement physique des informations générées par le logiciel sur des composants matériels.

Les diagrammes de déploiement sont constitués de plusieurs formes UML. Les boîtes en trois dimensions, appelées nœuds, représentent les composants du système, qu'ils soient logiciels ou matériels.

Les diagrammes de déploiement sont utiles dans plusieurs domaines, notamment pour :

- montrer quels éléments logiciels sont déployés par quels éléments matériels ;
- illustrer le traitement d'exécution du point de vue matériel ;
- visualiser la topologie du système matériel.

Qt Model Editor permet de réaliser des diagrammes de déploiement simples avec les éléments suivants :

- **Nœud** (*node*) : élément matériel ou logiciel représenté par une boîte en relief. Les ressources matérielles sont souvent représentées avec le stéréotype `<< device >>` ;
- **Composant** (*component*) : rectangle avec deux onglets et/ou le stéréotype `<<component>>` indiquant un élément logiciel ;

stereotype : **application**
stereotype display : **label**
plain shape : **no**



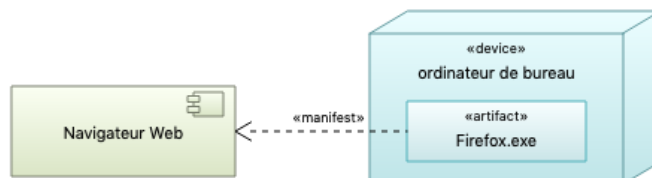
stereotype : **component**
stereotype display : **label**
plain shape : **yes**



stereotype : **component**
stereotype display : **decoration**
plain shape : **yes**



- **Artefact** (*artifact*) : produit développé par/pour le système, symbolisé par un rectangle stéréotype `« artifact »`. Un artefact qui est la manifestation d'un composant est relié à celui-ci par une relation de dépendance stéréotypé `<< manifest >>` orienté de l'artefact vers le composant ;

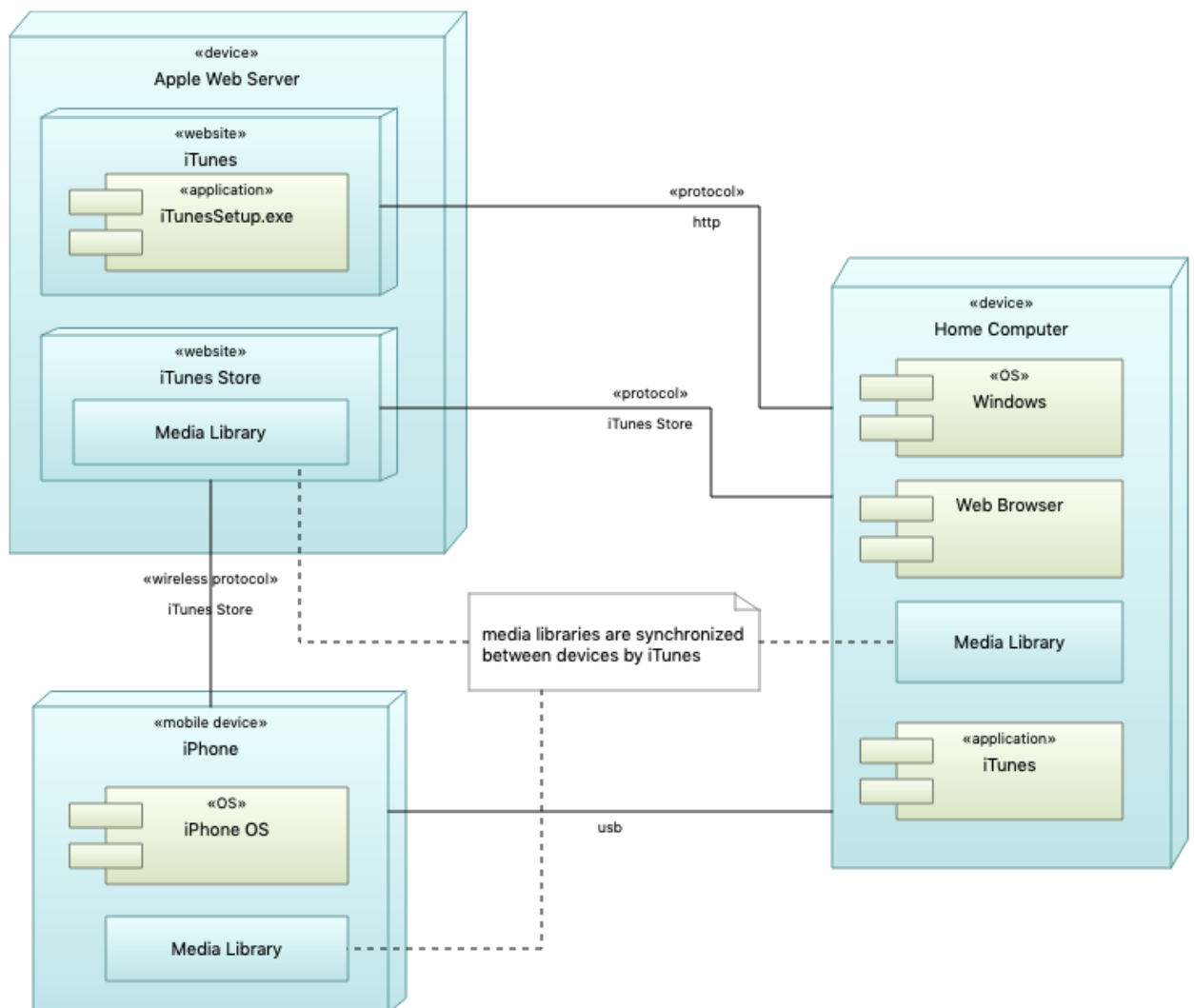


Note : La dépendance ci-dessus est dessinée à partir du composant, puis sa direction est inversée.

- **Association** : ligne continue indiquant un message ou tout autre type de communication entre deux nœuds. Le lien peut inclure des cardinalités et un stéréotype indiquant le type de liaison ;
- **Dépendance** : ligne pointillée terminée par une flèche, qui indique qu'un nœud ou composant est dépendant d'un autre.

Exemple : L'application de configuration iTunes peut être téléchargée à partir du site Web iTunes et installée sur un ordinateur personnel. Après l'installation et l'enregistrement, l'application iTunes peut communiquer avec Apple iTunes Store. Le client peut acheter et télécharger de la musique, des vidéos, des émissions de télévision, des livres audio, etc. et les stocker dans la médiathèque.

Les appareils mobiles comme Apple iPod Touch et Apple iPhone peuvent mettre à jour leurs propres bibliothèques multimédias à partir d'un ordinateur personnel avec iTunes via USB, ou peuvent télécharger des médias directement à partir d'Apple iTunes Store en utilisant un protocole sans fil.



Note : Le dessin de l'élément « noeud » est ici, pour des raisons purement esthétiques, modifié par rapport à la version d'origine...

... se reporter au chapitre « Personnalisation de l'éditeur de modèles ».

Les diagrammes de classes

Pour mémoire, une classe est une représentation abstraite d'un d'ensemble d'objets, elle contient les informations nécessaires à la construction de l'objet (c'est-à-dire la définition des attributs et des méthodes). La classe peut donc être considérée comme le modèle, le moule ou la notice qui va permettre la construction d'un objet. On dit qu'un objet est l'instance d'une classe (la concrétisation d'une classe).

Le diagramme de classes est un diagramme structurel (statique) qui permet de représenter :

- les **classes** (attributs + méthodes) ;
- les **associations** (relations) entre les classes.

Le diagramme de classes est le plus important des diagrammes UML, c'est le seul qui soit réellement obligatoire lors de la modélisation objet d'un système.

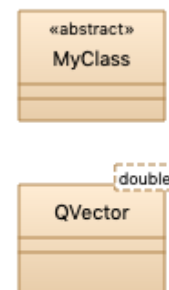
Représentation des classes

Une classe est représentée par un rectangle divisé en 3 compartiments :

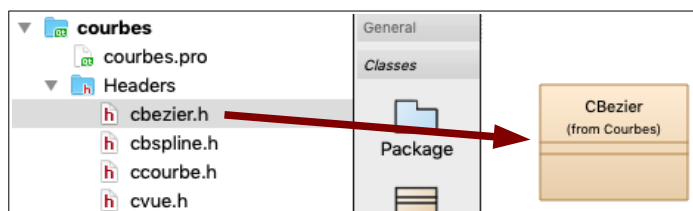
- le premier compartiment contient le **nom** de la classe (notation *Camel/Case*), il est en italique si la classe est abstraite (la classe peut aussi être stéréotypée <<abstract>>) ;
- le deuxième compartiment contient les **attributs** ;
- le troisième compartiment contient les **méthodes**.

En fonction des objectifs de modélisation, la représentation d'une classe peut être plus ou moins exhaustive. Lorsque la modélisation ne s'intéresse qu'aux relations entre les différentes classes du système, le renseignement des deux derniers compartiments est facultatif.

Pour les modèles de classes (*class template*), le paramètre en spécifié en haut à droite dans un cadre pointillé.



Avec Qt Creator, un fichier de définition (*header*) du projet logiciel en cours peut directement être glissé sur la zone de dessin du modèle en choisissant **Add class...** au moment de déposer.



Détails des membres

Chaque membre de classe peut être détaillé en précisant :

- le nom de l'attribut ou de la méthode ;
- la visibilité : public (+), protected (#) ou private (-) ;
- le type pour les attributs ;
- la signature pour les méthodes (liste des arguments avec leur type) ;
- le type de retour pour les méthodes.

Avec Qt Creator, les membres d'une classe peuvent directement être importés à partir de l'édition du fichier *header* par copier-coller vers la zone **Members** de la classe.

Qt framework oblige, les signaux sont préfixés par le symbole '>' et les slots par '\$'.

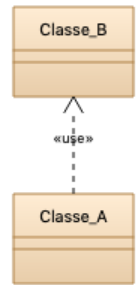
Relations entre classes

La barre d'outils attachée aux classes propose la mise en place de relations de type Dependency, Inheritance, Association, Aggregation et Composition.



La **dépendance** est la forme la plus faible de relation entre classes. Une dépendance entre deux classes signifie que l'une des deux utilise l'autre. Typiquement, il s'agit d'une relation transitoire, au sens où la première interagit brièvement avec la seconde sans conserver à terme de relation avec elle (liaison ponctuelle).

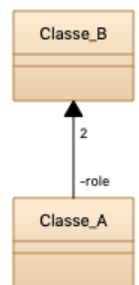
La dépendance est représentée par un trait discontinu orienté, reliant les deux classes. Elle est souvent stéréotypée « use » pour mieux expliciter le lien sémantique entre les éléments du modèle.



Alors que la dépendance autorise simplement une classe à utiliser des objets d'une autre classe, l'**association** signifie qu'une classe contiendra une référence (ou un pointeur) de l'objet de la classe associée sous la forme d'un attribut.

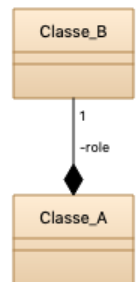
Cette relation est plus forte. Elle indique qu'une classe est en relation avec une autre pendant un certain laps de temps. La ligne de vie des deux objets concernés ne sont cependant pas associés étroitement (un objet peut être détruit sans que l'autre le soit nécessairement). La composition et l'agrégation sont des cas particuliers d'association.

L'association est représentée par un simple trait continu, reliant les deux classes. Les propriétés de navigabilité (*Navigable*) à chaque extrémité spécifient quelle classe possède une référence vers l'autre. L'association peut être enrichie par des rôles et des cardinalités.



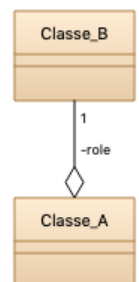
La **composition** indique qu'un objet A (appelé conteneur) est constitué d'un autre objet B. Cet objet B n'appartient qu'à l'objet A et ne peut pas être partagé avec un autre objet. C'est une relation très forte, si l'objet A meurt, alors l'objet B disparaît aussi. La composition se représente par un losange plein du côté de l'objet conteneur.

La manière habituelle d'implémenter la composition en C++ se fait soit directement au travers d'un attribut normal en utilisant la liste d'initialisation pour la création de l'objet composite, soit à l'aide d'une variable dynamique, auquel cas, ne pas oublier de rajouter un destructeur pour libérer cette variable dynamique.



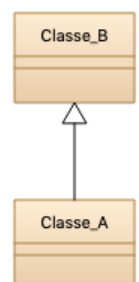
L'**agrégation** indique qu'un objet A possède un autre objet B, mais contrairement à la composition, l'objet B peut exister indépendamment de l'objet A. La suppression de l'objet A n'entraîne pas la suppression de l'objet B. L'agrégation se représente par un losange vide du côté de l'objet conteneur.

La manière habituelle d'implémenter l'agrégation en C++ se fait au travers d'un pointeur ou d'une référence qui pointe ou fait référence à l'objet agrégé déjà créé en dehors de l'objet conteneur (chaque objet a sa propre durée de vie).

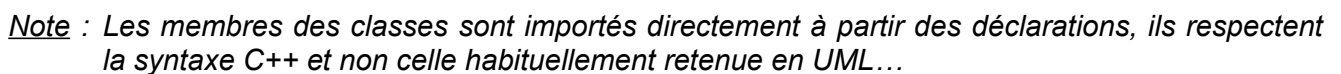


L'**héritage** indique qu'une classe A est une spécialisation d'une classe B. La classe A (appelé classe fille, classe dérivée ou sous-classe) hérite des attributs et des méthodes de la classe B (appelée classe mère, classe de base ou super classe). Les classes sont ainsi mises en relation par des caractéristiques communes (attributs et atomes de comportement) en respectant une certaine filiation.

L'héritage se représente par un triangle vide afin d'indiquer le sens de la généralisation (inverse de la spécialisation).



Qt 5.x Modeling - v1.0 - © avril 2020 - Alain MENU



The diagram shows four stages of the 'New Class' technique:

- 1. mise en place des éléments**: A 'New Class' box is shown next to a circle representing an item.
- 2. relation avec points de brisure**: The 'New Class' box is connected to the circle by a line with four square break points.
- 3. déplacement...**: The 'New Class' box is moved to the right, and the circle is moved to the left, with the break points following the movement.
- 4. ...jusqu'à cacher l'item**: The 'New Class' box is moved further to the right, and the circle is moved further to the left, such that the circle is now hidden behind the 'New Class' box.

```
classDiagram
    class Personne
    class "est enfant de"
    class "est frère/soeur de"
    class "est conjoint de"
    Personne "0..*" -- "2" "est enfant de"
    Personne "0..*" -- "1" "est frère/soeur de"
    Personne "1" -- "1" "est conjoint de"
```

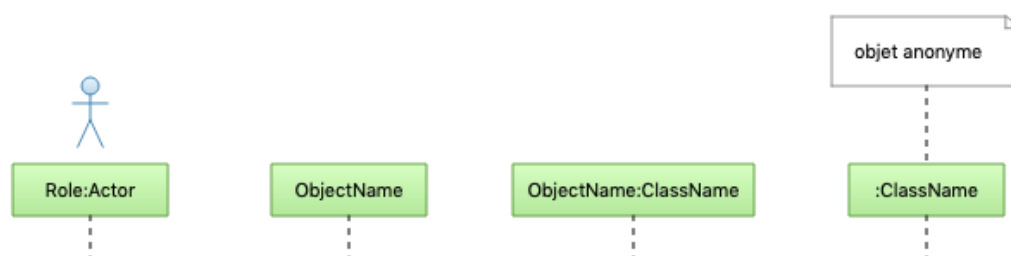
Les diagrammes de séquences

Le diagramme de séquence fait partie des diagrammes comportementaux (dynamique) et plus précisément des diagrammes d'interactions. Il permet de représenter des échanges entre les différents objets et acteurs du système en fonction du temps.

La modélisation d'un système nécessite en général un ensemble de diagrammes de séquences chacun correspondant à une fonctionnalité, généralement d'ailleurs pour illustrer un cas d'utilisation.

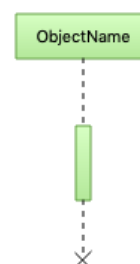
Représentation des objets

Dans un diagramme de séquence, l'**objet** (*instance*) ou l'**acteur** (*actor*) est représenté sous la forme d'un rectangle dans lequel figure le nom de l'objet ou le rôle de l'acteur. Le nom de l'objet est généralement souligné et peut prendre l'une des formes suivantes (le *stickman* est aussi considéré ici comme un objet mais Qt Model Editor ne prévoit pas de l'utiliser comme instance dans ce type de diagramme) :



Chaque objet est associé à une **ligne de vie** (en trait pointillés à la verticale de l'objet) qui peut être considérée comme un axe temporel (le temps s'écoule du haut vers le bas, mais sans quantification particulière).

La ligne de vie porte les **périodes d'activité** de l'objet (généralement, les moments où l'objet exécute une de ces méthodes). Une période d'activité est matérialisée par une sur-épaisseur de la ligne de vie. Lorsque l'objet est détruit, la ligne de vie s'achève par une croix.



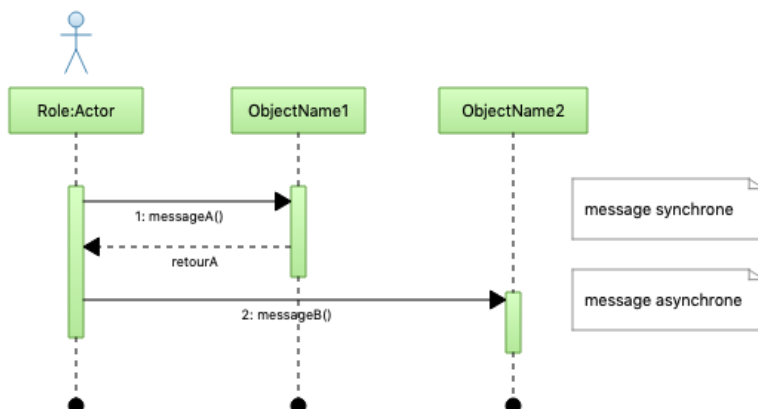
Les messages

Un **message** définit une communication particulière entre des lignes de vie, donc d'un objet vers un autre objet.

La réception d'un message est considérée par l'objet récepteur comme un événement qu'il faut traiter (ou pas). Plusieurs types de messages existent, les plus communs sont :

- l'invocation d'une opération : message **synchrone** d'appel d'une méthode de l'objet cible. L'expéditeur du message reste bloqué pendant toute l'exécution de la méthode et attend donc la fin de celle-ci et son éventuelle retour avant de pouvoir lancer un nouveau message ;
- l'envoi d'un signal : message **asynchrone** (typiquement utilisé pour la gestion événementielle). L'expéditeur n'attend pas la fin de l'activation de la méthode invoquée chez le destinataire ;
- la **création** ou la **destruction** d'une instance de classe au cours du cycle principal.

Un message porte en général le nom de la méthode ou du signal concerné, un message de retour porte souvent le nom de l'élément retourné.



Notes : Le tracé horizontal des messages est facilement contrôlable en ajoutant un point de brisure et en déplaçant celui-ci verticalement.

Les lignes de vie sont attachées en bas à des éléments Endlifeline.

Qt Model Editor place les noms des messages sous le trait, c'est dommage... L'outil ne prévoit pas non plus de numérotation automatique des messages.

Qt Model Editor ne prévoit pas de représentation spécifique pour les messages de retour...

... se reporter au chapitre « Personnalisation de l'éditeur de modèles ».

Cas des **messages récurrents** : Un objet peut s'envoyer un message à lui-même (utilisation d'une autre méthode du même objet). Cela se représente par un dédoublement de la bande d'activation ; un nouvel élément Activation est superposé à celui émetteur et deux points de brisure sont ajoutés au message pour le dessiner.



Les fragments d'interaction

Un **fragment d'interaction** est une partie du diagramme de séquence, délimitée par un rectangle et associée à une étiquette (dans le coin supérieur gauche). L'étiquette contient un opérateur d'interaction qui permet de décrire des modalités d'exécution des messages à l'intérieur du cadre.

Principaux fragments :

opérateur	opérande(s)	description
opt	[<condition>]	alternative partielle, fragment exécuté seulement si la condition est vraie
alt	[<condition>] [else]	alternative complète, les deux cas sont inscrits dans des cadres séparés (*)
alt	[<condition1>] [<condition2>] [else]	choix multiple, chaque cas est inscrit dans un cadre séparé (*)
loop	[<nombre>]	boucle exécutée <nombre> fois
loop	[<condition>]	boucle exécutée tant que <condition> est vraie

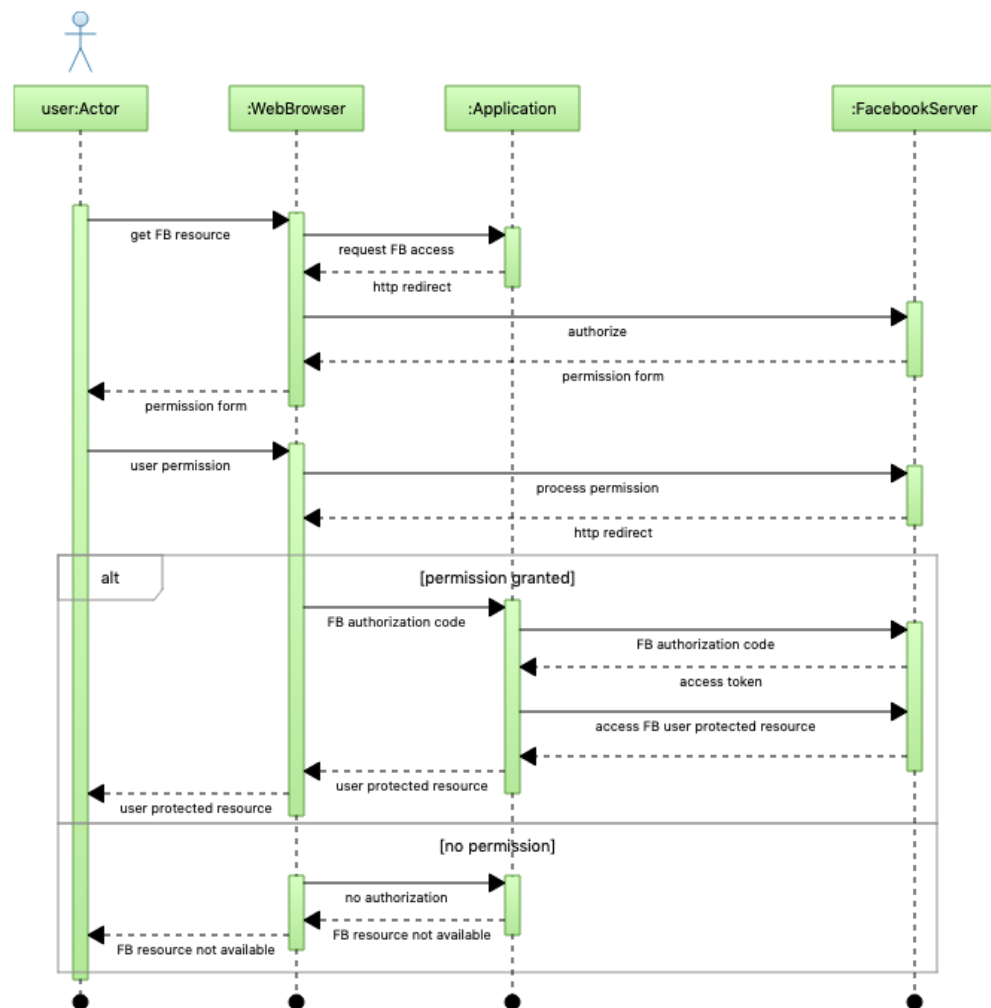
(*) les cadres superposés d'un même fragment sont normalement séparés par un trait pointillé.

Exemple : Facebook user authentication

Facebook utilise le protocole OAuth 2.0 qui permet à une application Web (appelée « client »), qui n'est généralement pas le propriétaire des ressources FB mais agit au nom de l'utilisateur FB, de demander l'accès aux ressources contrôlées par l'utilisateur FB et hébergées par le serveur FB. Au lieu d'utiliser les informations d'identification de l'utilisateur FB pour accéder aux ressources protégées, l'application Web obtient un jeton d'accès.

L'application Web doit être enregistrée par Facebook pour avoir un ID d'application. Lorsque la demande de certaines ressources Facebook protégées est reçue, le navigateur Web (« agent utilisateur ») est redirigé vers le serveur d'autorisation de Facebook avec l'ID d'application et l'URL vers laquelle l'utilisateur doit être redirigé après le processus d'autorisation.

L'utilisateur reçoit le formulaire de demande d'autorisation. Si l'utilisateur autorise l'application à obtenir ses données, le serveur d'autorisation Facebook redirige vers l'URI spécifié précédemment avec le code d'autorisation (« chaîne de vérification »). Le code d'autorisation peut être échangé par application Web contre un jeton d'accès OAuth.



Note : Qt Model Editor ne prévoit pas par défaut la représentation des fragments...
... se reporter au chapitre « Personnalisation de l'éditeur de modèles ».

Personnalisation de l'éditeur de modèles

Qt Model Editor ne permet pas forcément de représenter tous les éléments prévus par UML... Mais sa conception est telle qu'il est possible de personnaliser et d'étendre ses possibilités graphiques !

En effet, les éléments des palettes et les menus contextuels de relations sont intégralement définis dans un fichier *human readable* nommé **standard.def**.

Localisation :

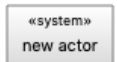
- Linux → /usr/share/qtcreator/modeeditor/standard.def
- MacOS → <QtinstallDir>/Qt Creator.app/Contents/Resources/modeeditor/standard.def
- Windows → <QtinstallDir>\Tools\QtCreator\share\qtcreator\modeeditor\standard.def

Le début du fichier présente la syntaxe reconnue par Qt Model Editor pour les éléments de types Icon, Relation, Dependency, Inheritance, Association et Toolbar. Viennent ensuite la définition des éléments pour les diagrammes de types Classes, Components, Use Cases, Activities, Sequences et Deployment. La fin du fichier concerne les barres d'outils de la palette.

Après en avoir fait une copie de sauvegarde, ce fichier peut être facilement modifié au moyen d'un éditeur de texte (p.e. Qt Creator :-), il suffit de redémarrer Qt Creator pour prendre en compte les modifications.

Diagrammes d'expression du besoin : acteur non humain

Création d'un nouvel élément dans la barre d'outils « UseCases » :



nouvelle définition dans la rubrique « Use Cases »...	menu de relations des éléments...
<pre>Icon { id: SystemActor elements: item name: "new actor" stereotype: 'system' display: label textAlignment: center width: 40 height: 30 baseColor: #a0a0a0 shape { Rect { x: 0; y: 0; width: 40.0; height: 30.0 } } }</pre>	<pre>Toolbar { id: UseCaseToolbar element: UseCase, Actor, SystemActor Tools { Tool { element: Communication } Tool { element: dependency } } }</pre>

barre d'outils...
<pre>Toolbar { id: UseCases Tools { Tool { title: "Package"; element: package } Tool { title: "UseCase"; element: item; stereotype: "usecase" } Tool { title: "Actor"; element: item; stereotype: "actor" } Tool { title: "SystemActor"; element: item; stereotype: "system" } Separator Tool { title: "Annotation"; element: annotation } Tool { title: "Boundary"; element: boundary } Tool { title: "Swimlane"; element: swimlane } } }</pre>

Le stéréotype « system » doit néanmoins être sélectionné dans les propriétés pour apparaître sur le dessin.



Diagramme de package : relation de communication avec les packages

Ajout d'une cible pour permettre les liaisons simples (trait continu) avec les paquetages :

modification de la relation « Communication » dans la rubrique « UseCases » :

```
Relation {
  id: Communication
  elements: UseCase, Actor, SystemActor, package
  pattern: solid
  color: A
  End {
    end: A
  }
  End {
    end: B
  }
}
```

Diagramme de cas d'utilisation : relations d'héritage

Ajout de relations d'héritage Actor vers Actor et UseCase vers UseCase (sans autoriser cette relation de Actor vers UseCase ni de UseCase vers Actor...) :



<i>nouvelle relation d'héritage Actor-Actor</i>	<i>nouvelle relation d'héritage UseCase- UseCase</i>
<pre>Relation { id: actorInheritance elements: Actor, SystemActor pattern: solid color: A End { end: A } End { end: B head: triangle } } Toolbar { id: UseCaseToolbar element: Actor, SystemActor Tools { Tool { element: actorInheritance } } }</pre>	<pre>Relation { id: usecaseInheritance elements: UseCase pattern: solid color: A End { end: A } End { end: B head: triangle } } Toolbar { id: UseCaseToolbar element: UseCase Tools { Tool { element: usecaseInheritance } } }</pre>

Nouvel item de type « note »

Création d'un cadre de note avec lien pointillé vers d'autres items, à utiliser conjointement avec l'élément Annotation qui gère la saisie multi-lignes de texte.



Le nouvel élément nommé NoteFrame peut être ajouté à la palette, par exemple dans les rubriques *UseCases*, *Class*, *Sequences*... (à défaut de pouvoir être simplement ajouté à *General*...). Exemple :

```
Toolbar {
  id: UseCases
  Tools {
    Tool { title: "Package"; element: package }
    Tool { title: "UseCase"; element: item; stereotype: "usecase" }
    Tool { title: "Actor"; element: item; stereotype: "actor" }
    Tool { title: "SystemActor"; element: item; stereotype: "system" }
    Separator
    Tool { title: "NoteFrame"; element: item; stereotype: "noteframe" }
    Tool { title: "Annotation"; element: annotation }
    Tool { title: "Boundary"; element: boundary }
    Tool { title: "Swimlane"; element: swimlane }
  }
}
```


Définition de l'élément NoteFrame, avec lien pointillé possible vers les cas d'utilisation et les acteurs :

nouvel élément...	nouvelle relation...
<pre> Icon { id: NoteFrame elements: item display: icon stereotype: 'noteframe' textAlignment: none width: 60.0 height: 30.0 lockSize: none baseColor: #c0c0c0 Shape { line { x0: 55.0; y0: 0.0; x1: 0.0; y1: 0.0 } line { x0: 0.0; y0: 0.0; x1: 0.0; y1: 30.0 } line { x0: 0.0; y0: 30.0; x1: 60.0; y1: 30.0 } line { x0: 60.0; y0: 30.0; x1: 60.0; y1: 5.0 } line { x0: 55.0; y0: 0.0; x1: 60.0; y1: 5.0 } line { x0: 55.0; y0: 5.0; x1: 60.0; y1: 5.0 } } } </pre>	<pre> Relation { id: NoteLink pattern: dash color: A End { end: A elements: NoteFrame } End { end: B elements: UseCase, Actor, SystemActor (*) head: none } } Toolbar { id: NoteFrameToolbar element: NoteFrame Tools { Tool { element: NoteLink } } } </pre>

(*) à étendre au fil des besoins :

UseCase, Actor, SystemActor, class, package, Node, component, Artifact, Instance

Modification d'aspect des éléments Node et NodeInstance

Diminution d'épaisseur de l'effet relief :

aspect d'origine...	nouvel aspect...
<pre> Icon { id: Node elements: item, package stereotype: 'node' display: icon textAlignment: top width: 20 height: 20 baseColor: #7ccad1 Shape { MoveTo { x: 0; y: 0 } LineTo { x: 3; y: -3 } LineTo { x: 23; y: -3 } LineTo { x: 23; y: 17 } LineTo { x: 20; y: 20 } LineTo { x: 0; y: 20 } Close Line { x0: 0; y0: 0; x1: 20; y1: 0 } Line { x0: 20; y0: 0; x1: 23; y1: -3 } Line { x0: 20; y0: 0; x1: 20; y1: 20 } } } </pre>	<pre> Icon { id: Node elements: item, package stereotype: 'node' display: icon textAlignment: top width: 20 height: 20 baseColor: #7ccad1 Shape { MoveTo { x: 0; y: 0 } LineTo { x: 1; y: -1 } LineTo { x: 21; y: -1 } LineTo { x: 21; y: 19 } LineTo { x: 20; y: 20 } LineTo { x: 0; y: 20 } Close Line { x0: 0; y0: 0; x1: 20; y1: 0 } Line { x0: 20; y0: 0; x1: 21; y1: -1 } Line { x0: 20; y0: 0; x1: 20; y1: 20 } } } </pre>

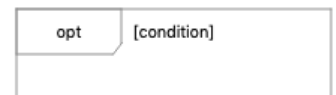
Messages de retour des diagrammes de séquences

Ajout d'une relation aux éléments de type Activation (rubrique *Sequences*) :

<i>nouvelle relation...</i>	<pre> Relation { id: ReturnMessage pattern: dash color: A End { end: A; elements: Activation } End { end: B; elements: Activation, Destruction; navigable: true; head: filledtriangle } }</pre>
<i>barre d'outils...</i>	<pre> Toolbar { id: ActivationToolbar element: Activation Tools { Tool { element: Lifeline } Tool { element: Message } Tool { element: ReturnMessage } } }</pre>

Fragments d'interaction

Ajout d'un élément FragmentSticker pour l'opérateur de fragment (opt, alt, loop...) et ajout d'un élément Fragment pour représenter le cadre du fragment et ses opérandes (à inscrire entre crochets) ; le FragmentSticker est à attacher en haut à gauche du Fragment :



<i>nouvel élément FragmentSticker...</i>	<i>nouvel élément Fragment...</i>
<pre> Icon { id: FragmentSticker elements: item display: icon stereotype: 'fragmentsticker' name: "opt" textAlignment: center width: 60 height: 20 minWidth: 60 minHeight: 20 lockSize: height baseColor: #c0c0c0 Shape { Line { x0: 0; y0: 0; x1: 60; y1: 0 } Line { x0: 60; y0: 0; x1: 60; y1: 15 } Line { x0: 60; y0: 15; x1: 55; y1: 20 } Line { x0: 55; y0: 20; x1: 0; y1: 20 } Line { x0: 0; y0: 20; x1: 0; y1: 0 } } }</pre>	<pre> Icon { id: Fragment elements: item display: icon stereotype: 'fragment' name: "[condition]" textAlignment: top width: 100 height: 50 lockSize: none baseColor: #c0c0c0 Shape { Line { x0: 0; y0: 0; x1: 100; y1: 0 } Line { x0: 100; y0: 0; x1: 100; y1: 50 } Line { x0: 100; y0: 50; x1: 0; y1: 50 } Line { x0: 0; y0: 50; x1: 0; y1: 0 } } }</pre>

<i>intégration des nouveaux éléments (rubrique <i>Sequences</i>)</i>	<pre> Toolbar { id: Sequences Tools { Tool { title: "Instance"; element: item; stereotype: "instance" } Tool { title: "Activation"; element: item; stereotype: "activation" } Tool { title: "Destruction"; element: item; stereotype: "destruction" } Tool { title: "EndLifeline"; element: item; stereotype: "endlifeline" } Tool { title: "Fragment"; element: item; stereotype: "fragment" } Tool { title: "FragmentSticker"; element: item; stereotype: "fragmentsticker" } Separator Tool { title: "NoteFrame"; element: item; stereotype: "noteframe" } Tool { title: "Annotation"; element: annotation } Tool { title: "Boundary"; element: boundary } Tool { title: "Swimlane"; element: swimlane } } }</pre>
--	--