Documentation utilisateur mascaret2

31 août 2011

F. Devillers et J. Soler

Un embryon de documentation utilisateur pour mascaret 2. Copyright ??? License ??

Table des matières

1 Introduction

1.1 But du document

Cette documentation s'adresse à l'utilisateur de `Mascaret 2`. L'utilisateur est la personne qui installe `Mascaret 2` sur une machine, puis crée des simulations. TODO

1.2 Qu'est-ce que Mascaret ?

TODO

1.3 Les plateformes

1.3.1 AReViMascaret

TODO

1.3.2 OgreMascaret

TODO

2 Installation

En l'état (31 août 2011), l'installation de `Mascaret 2` nécessite les connaissances suivantes: créer des répertoires, lancer des commandes dans un `shell`, recopier des fichiers, utiliser le gestionnaire de paquets (par exemple `synaptic`) sous `Ubuntu` ou `Windows`.

2.1 Structure, dépendances

La bibliothèque `Mascaret` est la partie commune à toutes les implémentations de `mascaret2`, et dépend de la `boost`, de `libxml2`. Une fois cette partie compilée nous obtiendrons le fichier `libMascaret.so` ou (`Mascaret.dll et Mascaretr.lib` sous Windows).

`AReViMascaret` dépend, comme son nom l'indique d'`ARéVi`, mais aussi d'`hlib2` et d'`animlib`.

Prévoyez donc un peu de temps pour installer tout cela. La suite de ce chapitre va vous donner des explications étape par étape. Les noms des paquets `Ubuntu` correspondant aux dépendances sont donnés quand ils existent.

2.2 Code source de `mascaret2`

Il est conseillé de commencer par créer un répertoire de travail qui contiendra le sous répertoire `mascaret2` a coté de sous répertoires pour certaines des dépendances.

Nous allons d'abord télécharger l'ensemble de `Mascaret 2`. Le code source est disponible sur le site `http://svn.cerv.fr/trac/mascaret2`. Le téléchargement se fait via l'outil `svn`. Si vous utilisez un `shell`, créez dans votre répertoire de travail un sous-répertoire `mascaret2` et déplacez vous y. La commande à lancer pour télécharger est précisée sur le site de `mascaret2`. Il vous faut un nom d'utilisateur et un mot de passe pour être autorisé à télécharger.

Une fois le téléchargement fini, vous devez obtenir l'arborescence suivante:

```
mascaret2
|-- C++
|   |-- AReViMascaret
|   |-- Datas
|   |-- HTTPServerBaseDir
|   |-- IrrlichtMascaret
|   |-- OgreMascaret
|   |-- Mascaret
|   `-- WrapperBoostPy
|-- Ext
|-- GeneratedModelDoc
|     ...
|-- Model
|     ...
|-- Publication
|     ...
|-- Python
|     ...
```

```
'-- Tools
      ...
```

Les répertoires qui vont nous intéresser particulièrement sont `mascaret2/C++/Mascaret`, `mascaret2/C++/AReViMascaret` et `mascaret2/C++/OgreMascaret`. Il est donc possible de ne récupérer que le répertoire C++. Les commande serait alors: `svn co --username=[login] --no-auth-cache http://svn.cerv.fr/svn/mascaret2/trunk/C++/`. Nous allons commencer par la compilation de `Mascaret`, puisqu'`AReViMascaret` en dépend.

2.3 Dépendances et compilation de `Mascaret` sous Linux

`Mascaret` dépend de la `boost` et de `libxml2`. Il y a de grandes chances pour que `libxml2` soit déjà installée. Vous pouvez avoir besoin du paquet `libxml2-dev`.

Pour le compiler vous allez avoir besoin de `g++` qui n'est pas installé par défaut sur `Ubuntu`. Installez le paquet `g++` si nécessaire. La compilation se fait via un script python générant un makefile puis par l'outil `make`. Ce procédé est le même pour l'ensemble des projets.

Les portions de la `boost` utilisées correspondent aux paquets: `libboost-filesystem-dev libboost-python-dev libboost-dev libboost-test-dev libboost-signals-dev`. Si vous n'avez pas accès aux paquets, le site de la `boost` est http://www.boost.org.

Maintenant que les dépendances sont intallées, placez vous dans le répertoire `mascaret2/C++/Mascaret` et lancez la commande `./configure`. Si tout se passe bien un fichier `makefile` est généré. Vous pouvez alors lancer la commande `make`. Pensez à utiliser l'option `-j` si vous avez un processeur multi-core (`make -j3` pour un core duo et ainsi de suite). A la fin de la compilation vous obtenez la bibliothèque `libMascaret.so`.

2.4 Dépendances et compilation d'`AReViMascaret` sous Linux

En plus de `Mascaret`, `AReViMascaret` dépend d'`ARéVi` de `hLib2` et d'`AnimLib`.

`ARéVi` est disponible sur le site http://svn.cerv.fr/trac/AReVi/. Une documentation d'installation y est fournit pour son installation. Il vous faudra récupérer pour le moment la branche principale d'`ARéVi`.

`hlib2` est une bibliothèque d'animation d'humanoïde pour `ARéVi`, elle est disponible à partir du site http://svn.cerv.fr/trac/hLib. La récupération des sources se fait comme indiqué sur le site, avec `svn`. Vous n'avez cependant pas besoin d'un compte et d'un mot de passe, `guest` suffit ici. Son installation se fait comme ARéVi avec l'aide de l'outil `scons`. Les commandes sont `scons` pour compiler. `scons install` pour installer.

`AnimLib` est une bibliothèque d'animation de solides à partir de NURBS pour ARéVi. Le projet est disponible à l'adresse http://svn.cerv.fr/trac/AnimLib. Comme pour `ARéVi` et `hLib`, il se compile et s'installe à l'aide de `scons`

Nous pouvons maintenant passer à la compilation d'`AReViMascaret`. Pour compiler, placez vous dans le répertoire `mascaret2/C++/AReViMascaret` et lancez les commandes `./configure` pour générer le makefile. `make` pour compiler

2.5 Dépendances et compilation de OgreMascaret sous Linux

OgreMascaret dépends de la bibliothèque 3D Ogre en version 1.7.2 minimum. Le projet Ogre3D est disponible à l'adresse http://www.ogre3d.org/. Les packets disponibles sur Ubuntu sont trop anciens. Il vous faut donc télécharger les sources, les compiler et les installer. La documentation est fournit dans l'archive téléchargée.

2.6 Dépendances et compilation de Mascaret sous Windows

TODO

2.7 Dépendances et compilation de AReViMascaret sous Windows

TODO

2.8 Dépendances et compilation de AReViMascaret sous Windows

TODO

3 Les applications Mascaret

3.1 Structure d'une application

Une application Mascaret est composé de:

- Un modèle UML exporté au format XMI (1.4/2.0/2.1)
- La description d'un environnement instanciant le précédent modèle dans un format spécifique à Mascaret (fichier(s) XML).
- Un fichier configurant la Navigation, les interactions et le rendu (fichier XML)

3.2 Le modèle

Le modèle permet de décrire les types d'objets/personnes qui vont évoluer dans la simulation. Il va permettre par exemple de décrire la composition d'un porte-avion, d'un avion, d'un camion de pompier ou même d'une prise électrique programmable ainsi que les rôles des divers acteurs de la simulation.

3.2.1 Les outils de modélisation UML

Les modèles UML utilisé par Mascaret peuvent être réalisé par n'importe quel modeleur UML pouvant exporté dans un format XMI surporté par Mascaret (1.4/2.0/2.1). Les modeleurs avec lesquels des applications Mascaret ont été réalisées sont :

- BOUML: Un modeleur UML open source très complet
- Objecteering: Un modeleur propriétaire assurant la cohérence du modèle
- Modelio: Le successeur d'objecteering dont la version gratuite supporte l'export en XMI

3.2.2 Le profile Mascaret

Un profile UML Mascaret est disponible pour objecteering. Pour les autres modeleur, il convient d'ajouter au modèle les types de base suivants:

- `integer` représentant un entier naturel
- `real` représentant un nombre réel
- `string` représentant une chaine de caractère
- `boolean` représentant une valeur booléenne (vrai ou fausse)
- `vector` représentant un vecteur de réel à 3 dimensions
- `rotation` représentant une orientation dans l'espace définit par un vecteur à 3 dimensions et un angle
- `point` représentant un point orienté dans l'espace
- `path` représentant une suite ordonnée de points orientés dans l'espace
- `color` représentant une couleur
- `shape` représentant une représentation 3d
- `animation` représentant un animation
- `sound` représentant un son.

Il n'est toutefois pas nécessaire de spécifier tous ces type de base. Seuls ceux utilisé dans le modèle sont bien entendu obligatoire.

De même, les stéréotypes suivants sont nécessaires:

- `Entity` applicable sur une classe. Les classes de ce stéréotype sont des entités. Elles possèdent implicitement une position dans l'espace et une propriété `shape` de type `shape` (voir ci-dessus). Elles peuvent de plus avoir une entité parentes et des entités enfantes.
- `Agent` applicable sur une classe (pas utilisé pour le moment). Les classes de ce stéréotype sont des agents. Ils peuvent posséder des comportements d'agent.
- `VirtualHuman` applicable sur une classe. Les classes de ce stéréotype sont à la fois des entités et des agents.
- `RoleClass` applicable sur une classe. Les classes de ce stéréotype représente un rôle au sein d'une équipe.
- `Team` applicable sur un package. Les packages de ce stéréotype sont des équipes. Ces packages contiennent généralement des classes stéréotypés `RoleClass` et des activités stéréotypé `Procedure`
- `Procedure` applicable sur un diagramme d'activité. Les activités de ce stéréotype sont des procédures devant être réalisés par les équipes.
- `NonInterupt` applicable sur un état de machine à état. Les états de ce stéréotype sont non interuptible. C'est à dire que les actions réalisés sur leur `do` doivent être réalisé jusqu'au bout même si une transition devient valide pour quitter cette état.

3.2.3 Le méta-modèle de Mascaret

Mascaret implémente un métamodèle proche du métamodèle UML 2. Cependant, quelques différences existent. Un modèle est constitué d'un unique package racine.

Un package peut contenir:

- des packages
- des classes
- des signaux

Une classe peut contenir:

- une unique classe mère
- des propriétés (property dans UML2) pouvant être des attributs (type de base) et des associations. Les différents type d'associations comme compositions, agrégations sont considérer comme des associations simples.
- des méthodes
- des machine à états

Une méthode est composé de:

- paramètres
- un type de retour
- une éventuelle activité fournissant son implémentation

Une machine à état est composé de:

- états possédant éventuellement un comportement lors de l'entrée, un lors de la sortie et un lorsque l'état est actif.
- des transitions pouvant contenir un trigger (un signal validant la transition)

Les Region, History et SuperState du modèle UML2 ne sont pas gérées par Mascaret.

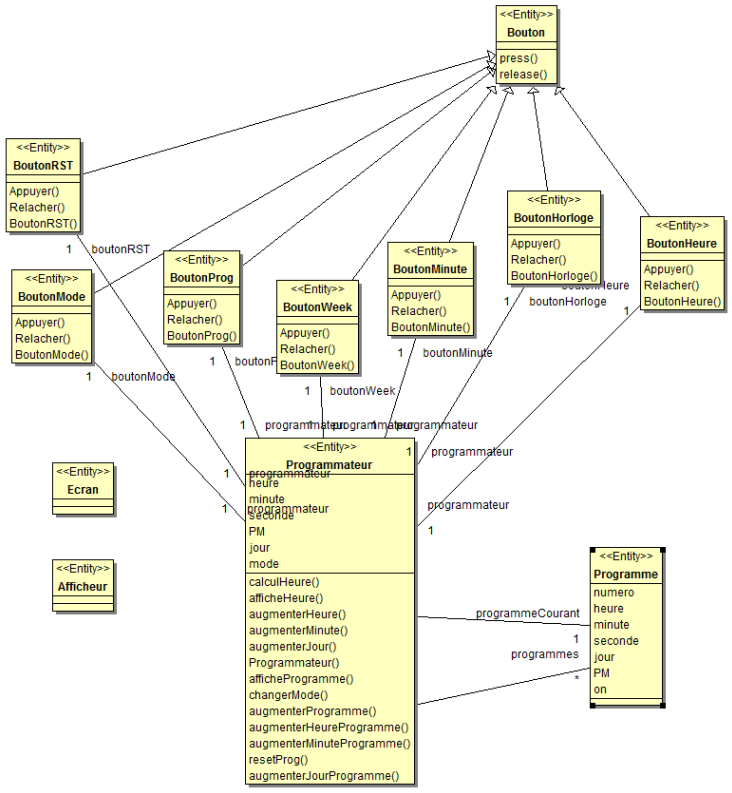
Une activité est composée de:

- Action pouvant être du type CallOperation et SendSignal
- des transitions pouvant contenir une condition en OCL (voir ci dessous)

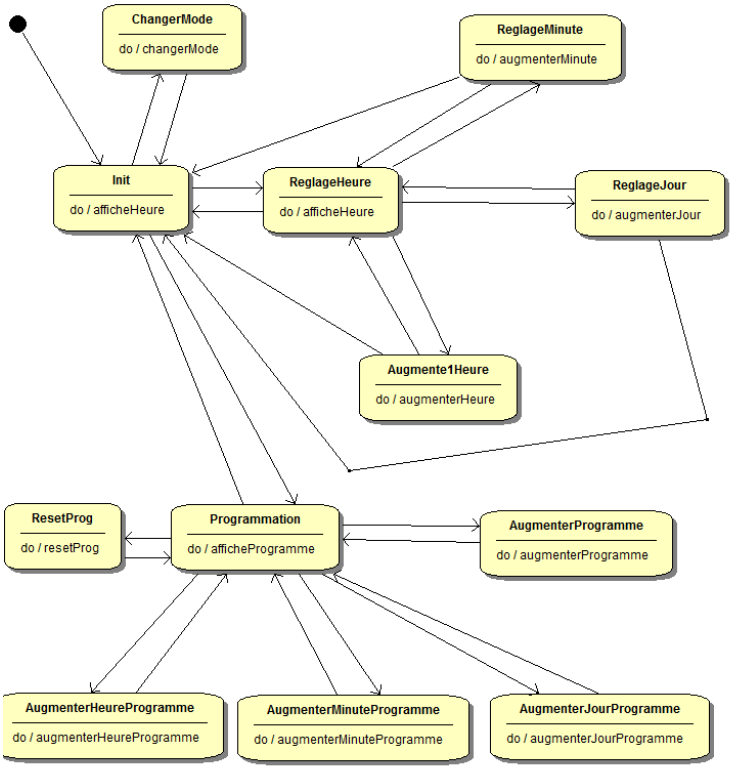
3.2.4 Le langage OCL

Les transitions des activités peuvent posséder une condition. Ces conditions sont exprimées dans le langage OCL. TODO

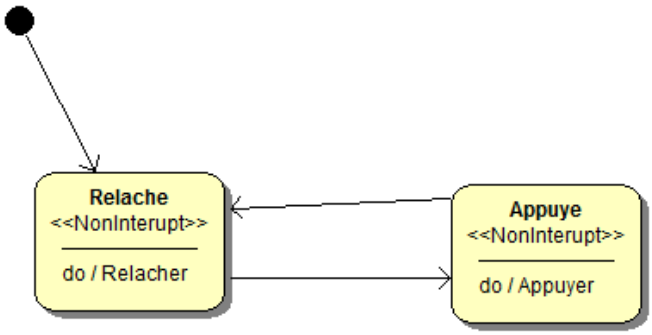
3.2.5 Exem



Cette image présente la partie statique du modèle d'une prise électrique programmable. Cette prise possède 8 boutons et plusieurs programmes plus un courant.

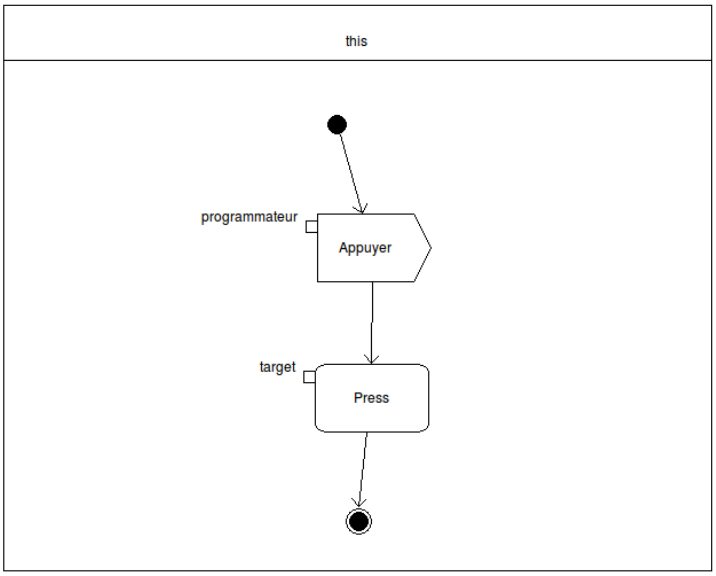


Les comportements de cette prise programmable est décrit par la machine à état ci-dessus. Cette machine à état étant déclarée dans la classe `Programmateur`, toute instance de Programmateur dans l'environnement exécutera dès sa création une instance de cette machine à état. La machine à état commencera par activé l'état `Init` et donc à appelé la méthode `afficheHeure`. Bien que ce ne soit pas visible sur le diagramme, les transitions sont liées à des triggers. C'est à dire qu'elles deviennent valides lorque l'entité reçoit un signal correspodant au trigger. Par exemple, la transition allant de l'état `Init` vers l'état `ReglageHeure` est valide lorsque que l'instance du Programmateur reçoit le signal `boutonHorlogeAppuye`.



La machine à état ci-dessus présente le comportement d'un des boutons. La transition de `Relache` vers `Appuye` attend le signal `Appuye`. La transition de `Appuye` vers `Relache` attend le signal `Relache`. On constate que les opérations réalisées dans les 2 états ne

sont pas interruptibles. C'est à dire que lorsque l'on est dans l'état `Relache` par exemple, la méthode `Relacher` doit être terminé pour que l'envoi d'un signal `Appuye` entraine le changement d'état.



Le diagramme d'activité ci-dessus décrit le comportement de la méthode `Appuyer` d'un des boutons. Elle consiste à envoyer un signal `"boutonXXXAppuye"` (le signal dépend du type de bouton) la cible programmateur (il s'agit ici d'un attribut de la classe du bouton mais cela aurait pu être un argument de la méthode) puis à executer la méthode `press` de la classe mère `Bouton`. Les méthodes `Relacher` des boutons sont implémentées de la même façon (signal `boutonXXXRelache` au lieu de `boutonXXXAppuye` et `release` au lieu de `press`).

Ainsi, avec l'ensemble des diagrammes spécifier dans le modèle, on peut déduire le comportement d'une prise programmable:

1. Lors de l'appuie sur un bouton (par exemple le bouton Horloge), le signal appuie sera envoyé au bouton Horloge. Ce comportement sera explique dans plus tard (voir Les interactions)
2. La machine à état du bouton Horloge reçoit le signal et active l'état `Appuye` lançant la méthode `Appuyer` de la classe du bouton (ici `BoutonHorloge`).
3. L'activité implémentant la méthode `Appuyer` de la classe `BoutonHorloge` s'exécute en envoyant le signal `boutonHorlogeAppuye` au programmateur et en lançant la méthode `press` de la classe `Bouton` (comportement opaque, voir Les comportements opaques).
4. Le programmateur reçoit le signal `boutonHorlogeAppuye` et active l'état `ReglageHeure` lançant la méthode `afficheHeure`.

Ensuite, une fois dans cette état, l'appui sur le bouton heure par exemple déclenche une suite d'action sur le même principe finissant par activer l'état `Augmente1Heure` lançant la méthode `augmenterHeure` (comportement opaque) augmentant concrètement l'heure courante du programmateur de 1 heure. La transition entre `Augmente1Heure` et `ReglageHeure` ne possédant pas de trigger, l'état `ReglageHeure` est tout de suite réactiver relançant la méthode `afficheHeure`

3.3 L'environnement

L'environnement est une instanciation du modèle. Il s'agit d'un ensemble d'instance des classes définit dans le modèle.

3.3.1 Structure des fichiers de description de l'environnement

Le fichier environnement est un fichier XML contenant la description de l'ensemble des instances. Son format est propriétaire. La racine du fichier environnement est une zone comprenant l'ensemble des entités de l'environnement. Il est préférable qu'un environnement référence sont modèle (pour l'instant (31 août 2011), cela ne sert à rien puisque la déclaration du modèle est réaliser dans le fichier de configuration mais ce comportement devrait changer). Un fichier environnement est donc structuré ainsi:

```
<?xml version="1.0"?>
<Area>
        <Model url="./model.xmi"/>
        <!-- Contenu de l'environnement -->
</Area>
```

3.3.1.1 Description des entités

Une entité se décrit à l'aide d'une balise `Entity`. Une entité peut contenir une description de sa position dans l'espace via des balises `Position` et `Orientation`. Elle peut de plus contenir des représentations graphique via les balises `Shape` ainsi des valeurs pour ces attributs via les balises `Attribute`. Enfin, on peut renseigner des animations grâce à la balise `Animation` et des sons grâce à la balise `Sound`. Par exemple, on pourra décrire une prise programmable comme ceci:

```
<Entity name="Prise" class="Programmateur">
        <Position x="0" y="0" z="0" />
        <Orientation roll="0" pitch="0" yaw="0" />
        <!--Il est aussi possible de définir l'orientation ainsi: /-->
        <!--Orientation x="1" y="0" z="0" angle="3.1415" /-->
        <Shape url="VRMLS/Prise.wrl"/> <!-- Si l'attribut name n'est pas renseigné, on pren
        <Entity name="p01" class="Programme">
                <Attribute name="numero" value="1" />
                <Attribute name="on" value="true" />
        </Entity>
        <Entity name="p02" class="Programme">
                <Attribute name="numero" value="1" />
                <Attribute name="on" value="false" />
        </Entity>
        <!-- Autres programmes -->
        <Attribute name="boutonHeure" value="BoutonHeure"/>
        <Attribute name="boutonMinute" value="BoutonMinute"/>
        <!-- Affectations des autres boutons -->
        <Attribute name="programmes" value="p01" />
        <Attribute name="programmes" value="p02" />
        <-- Autres instances de programmes -->
</Entity>
```

On remarquera ici les balises `Entity` déclarant les entités filles "p01" et "p02". Ces entités n'ont pas de représentations graphique (balise `Shape`) mais si elles en avait une, les positions et orientations serait décrites dans le repère de l'entité parent. De plus, les entités filles sont "attachées" à leur entité parente. Si l'entité parente se déplace, les entités filles se déplacent de la même façon.

3.3.1.2 Description des zones

Une zones est plus ou moins similaire à une entité. Elle me fait que référencer les entités quelle contient. Elles se déclare grâce à la balise `Area`.

3.3.1.3 Description humain virtuels

Les humains virtuels se déclare grâce à la balise `VirtualHuman` Les humains virtuels possèdent les mêmes attributs que les entités avec en plus: les comportements d'agent grâce à la balise `Behavior` les base de connaissance grâce à la balise `KnowledgeBase` (Cette balise est déconseillée) Pour une description des comportements d'agents, voir Les comportements d'agents Par exemple:

```
<VirtualHuman name="Paul" class="Professeur">
        <Position x="10.27" y="5.32" z="0" />
        <Orientation roll="0" pitch="0" yaw="0" />
        <Shape url="Professeur.wrl"/>
        <Attribute name="firstname" value="Paul"/>
        <Attribute name="subject" value="physique"/>
        <Behavior name="SimpleCommunicationBehavior" />
        <Behavior name="SuperviseLearner" />
</VirtualHuman>
```

3.3.1.4 Description de points de vue

Il est possible de définir des points de vue grâce à la balise `Viewpoint`. Les points de vues sont simplements des points orientés dans l'espace où la caméra peut se déplacer. Ils doivent contenir les balises `Position` et `Orientation`.$* Par exemple:

```
<Viewpoint name="Camera">
        <Position x="0" y="0.17" z="0.8" />
        <Orientation roll="-0.979" pitch="1.57" yaw="0.584" />
</Viewpoint>
```

3.3.1.5 Description de points orientés

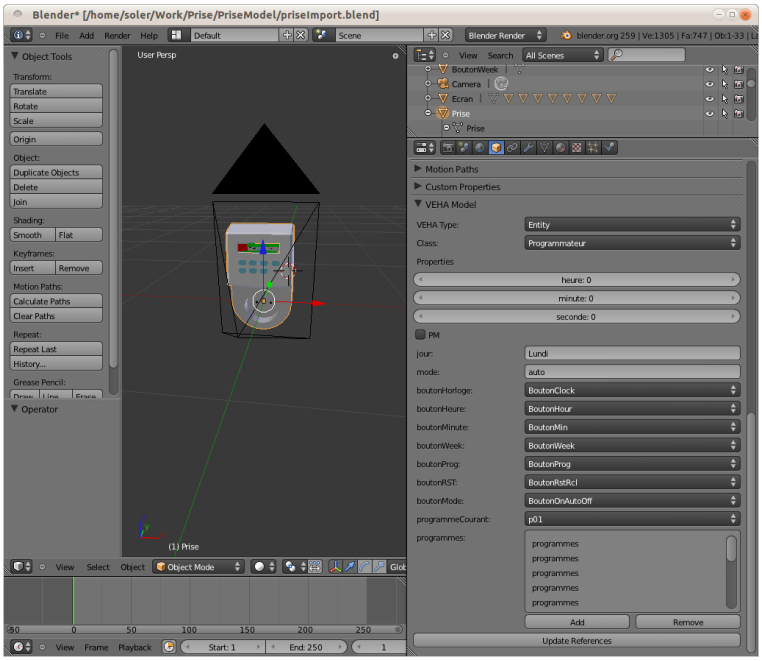
Il est possible de définir des points orientés à la balise `Topological`. Les points orientés peuvent servir à de nombreuses applications comme la description de chemin dans l'environnement, la définition de points de préhension d'un objet... Ils doivent contenir les balises `Position` et `Orientation`.$* Par exemple:

```
<Topological name="MonPoint">
        <Position x="0" y="0.17" z="0.8" />
        <Orientation roll="-0.979" pitch="1.57" yaw="0.584" />
</Topological>
```

3.3.1.6 Description des organisations

TODO

3.3.2 L'éditeur d'environnement Blender

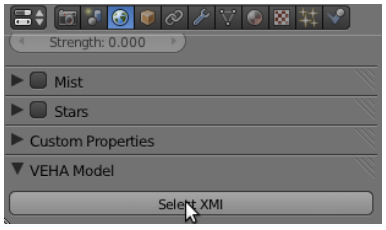


Le répertoire Plugins de Mascaret contient un répertoire BlenderEditor contenant un plugin pour Blender permettant l'édition d'un environnement Mascaret.

3.3.2.1 Installation

Pour installer ce plugin, il faut créer un lien symbolique ou copier le répertoire `veha_blender_editor` dans le répertoire `blender/2.5x/scripts/addons`. Il existe quelques problème sur les scripts d'import/export de fichiers vrml dans blender 2.58 et 2.59 (dernière version en date à ce jour). Les scripts corrigés sont fournit dans le répertoire BlenderEditor et sont à copier dans le répertoire `blender/2.5x/scripts/addons/io_scene_x3d`. Il faut ensuite l'activer en allant dans le menu "File > User preferences", onglet Add-Ons et en cochant "Import-Export: VEHA Importer/Exporter". Vous pouvez ensuite cliquer sur "Save As default" pour que le plugin soit toujours activer.

3.3.2.2 Utilisation



Pour que le plugin soit utile, il faut chager un modèle. Pour cela, allez dans l'onglet World de la fenêtre des propriétés (une icone avec une terre). Tout en bas, une section VEHA Model vous propose de choisir un fichier XMI.

Une fois le modèle charger, il est possible de réaliser un import d'un environnement existant. Pour cela, allez dans le menu "File > Import > Import VEHA Environment". Une boîte de dialogue de choix de fichier vous permet de choisir le fichier à importer. Des options d'import sont disponibles sur la gauche:

- Redo VRML rotation: Les exporters VRML applique une rotation de -PI/2 a tous l'environnement. L'importer supprime cette rotation. Mais dans AReVi, le fichier est lu tel quel. L'exporter VEHA compense donc cette rotation. Mais si l'environnement à importer n'a pas été réaliser avec l'exporteur VEHA, si l'importeur VRML supprime cette rotation, la scene n'est plus cohérente... Cette option permet de refaire la rotation VRML supprimer par l'importeur.
- Build vertex groups: Chaque fichier VRML sera considérer après import comme un seul et unique objet Blender (on fait un join sur tous les objets importer du VRML). Cette option permet de créer un vertex group par objet importer dans un VRML (l'import sera plus long).
- Temporary Layer: Le script d'import à besoin de disposer d'un calque blender vide. Cette option permet de choisir le calque temporaire à utiliser (de 0 à 19).

L'export se réalise en allant dans le menu "File > Export > Export VEHA Environment". La boîte de dialogue de choix de fichier présente une option sur la gauche: Export Type. Cette option permet de choisir un export En VRML (pour la plateform AReVi ou en Ogre. Pour l'export Ogre, vous devez disposer du plugin Blender2Ogre (http://code.google.com/p/blender2ogre/). Cependant, ce plugin evolue souvent. Depuis la version 0.5, il n'est plus possible d'exporter dans un système de coordonné différent de celui de Ogre.

Pour éditer une entité de l'environnement, il faut aller dans l'onglet object de la fenêtre de propriété de blender. Une section VEHA Model permet d'editer la classe de l'objet selectionné ainsi que de renseigner l'ensemble de ses propriétés. Un bouton "Update References" permet de mettre à jour les listes de références vers d'autres objet.

3.3.3 L'editeur d'environnement 3DS Max

Le répertoire Plugin de Mascaret comporte un répertoire 3dsEditor contenant un plugin 3DS Max pour editer un environnement Mascaret. Ce plugin est en cours de développement et est pour l'instant loin d'être vraiment utilisable en l'état. TODO

3.3.4 Exemple d'environnement

Un exemple complet de la description de l'environnement d'une prise électrique programmable se trouve en Annexe (voir Chapitre 4 [Annexe1], page 22)

3.4 Les plugins

3.4.1 Les évènements

Les évènements Mascaret sont des fonctions `extern "C"` pouvant être présentent dans chaque plugin. Ces fonctions sont:

- MASCARET_INIT: Cette fonction est appelée à l'ouverture du plugin. Ni le modèle, ni l'environnement ne sont alors chargés. Elle peut par exemple servir à initialiser une bibliothèque.
- MASCARET_MODEL_LOADED: Cette fonction est appelée lorsque le modèle et l'environnement sont chargé. Elle peut par exemple servir à modifier l'environnement au début de la simulation.
- MASCARET_STEP: Cette fonction est appelée à chaque pas de temps de la simulation. Elle peut par exemple permettre la mise à jour des données d'un périphérique ou lancé un comportement spécifique indépendant de Mascaret.
- MASCARET_DESTROYING: Cette fonction est appelée à la fin de la simulation. Elle permet généralement de nettoyés les ressources utilisées dans le plugin.

3.4.2 Les comportements opaques

Dans le modèle, lorsqu'une méthode déclarée pour une classe ne possède pas d'implémentation par une activité, cette méthode doit être implémenté par un comportement opaque. Il s'agit en fait d'implémenter une classe abstraite de Mascaret: `BehaviorExecution`. Cette classe possèdent une méthode virtuel pur `double execute()` à surdéfinir. La valeur de retour de la méthode `execute` correspond au temps en second au bout duquel la méthode doit être rappelée. Un retour de 0 correspond à la fin de l'execution de la méthode. Enfin, Mascaret pour instancier ce comportement va chercher une fonction:

```
extern "C" BehaviorExecution* NomDeLaClasse_NomDeLaMethode_init(shared_
ptr<Behavior> specif,shared_ptr<InstanceSpecification> host, const
Parameters& p);.
```

Les paramètres de cette fonction correspondent au paramètres du constructeur de la classe `BehaviorExecution`. Le paramètre `specif` contiendra la méthode dans le méta-

modèle Mascaret. Le paramètre `host` contiendra l'instance de la classe. Le paramètre `p` contiendra les paramètres d'appel de la méthode. Pour implémenter un comportement, on utilise la bibliothèque Mascaret et/ou les bibliothèque AReViMascaret et OgreMascaret. Si l'une de ces 2 dernières bibliothèques est utilisée, le plugin ne sera utilisable que sur la plateforme correspondante. Une documentation de ces bibliothèque peut être générée avec doxygen (http://www.stack.nl/~dimitri/doxygen/). Voici un exemple de code d'un comportement opaque:

```
#include "VEHA/Behavior/Common/BehaviorExecution.h"
#include "VEHA/Behavior/Common/Behavior.h"
#include "VEHA/Entity/Entity.h"
using namespace VEHA;

class Bouton_press : public BehaviorExecution
{
public:
          Bouton_press(shared_ptr<Behavior> specif,
                              shared_ptr<InstanceSpecification> host,
                              const Parameters& p)
          : BehaviorExecution(specif, host, p)
          {
          }
          virtual ~Bouton_press()
          {
          }
          double execute()
          {
                    VEHA::Vector3 b = (shared_dynamic_cast<VEHA::Entity>(getHost()))->getLocalPosition();
                    b.z -= 0.007;
                    (shared_dynamic_cast<VEHA::Entity>(getHost()))->setLocalPosition(b);
                    return 0;
          }
};
VEHA_PLUGIN BehaviorExecution* Bouton_press_init(shared_ptr<Behavior> specif,
                                                  shared_ptr<InstanceSpecification> host,
                                                  const Parameters& p)
{
          return new Bouton_press(specif, host,p);
}
```

3.4.3 Les comportements d'agents

Pour implémenter un comportement d'agent, il faut dérivé d'une de ces classes:

- `SimpleBehaviorExecution`: exécute la méthode à surdéfinir `void action()` à interval régulier (attribut interval) tant que la méthode à surdéfinir `bool done()` renvoit `true`.
- `CyclicBehaviorExecution`: exécute la méthode à surdéfinir `void action()` de façon cyclique tant que l'interval n'est pas null.
- `OneShotBehaviorExecution`: exécute la méthode à surdéfinir `void action()` une seule fois seulement.

3.4.4 Les base de connaissances (déconseillé)

TODO

3.4.5 Les plugins standards

Le répertoire Plugin dans Mascaret contient des plugins fournit en standard et pouvant être utilisé dans n'importe quelle simulation.

3.4.5.1 SpaceMouse

Ce plugin permet d'utiliser une souris 3D (de marque 3D connexion). Elle déclare dans la liste des péripheriques de l'application un péripherique "spaceMouse" contenant: TODO

3.4.5.2 TouchScreen

Ce plugin permet d'utiliser un écran tactile (de marque ??). Elle déclare dans la liste des péripheriques de l'application un péripherique "touchScreen" contenant: TODO

3.4.5.3 VirtualHumanBehaviors

TODO

3.5 La configuration de la simulation

La configuration de la simulation se fait dans un fichier xml. C'est le fichier d'entrée de l'application. En effet, lorsqu'on lance une simulation avec ogreMascaret ou areviMascaret, c'est ce fichier que l'on passe en paramètre. Par convention, nous nommons ce fichier avec l'extension ".mas" afin de le differencier des autres fichiers de la simulation.

3.5.1 Structure de base

Ce fichier doit donc référencer le modèle, l'environnnement, les acteurs, les organisations et doit contenir les éléments configuration de la simulation. La balise racine de ce fichier est une balise `Application`. Ces balises filles sont:

- `Model`: l'attribut url de cette balise définit le chemin relatif a ce fichier vers le fichier model (.xmi)
- `Environment`: l'attribut url de cette balise définit le chemin relatif a ce fichier vers le fichier environnement
- `Actors`: l'attribut url de cette balise définit le chemin relatif a ce fichier vers le fichier acteurs
- `Organisations`: l'attribut url de cette balise définit le chemin relatif a ce fichier vers le fichier organisations
- `Navigation`: définissant les paramètres de navigation
- `Interactions`: définissant les interactions
- `Scene`: définissant les paramètres de la scène (skybox, lumières, ...)
- `Renderer`: définissant les paramètres de la fenêtre de rendu

```
<?xml version="1.0"?>
<Application>
          <Model url="./programmateur.xmi" />
          <Environment url="./Prise.xml" />
          <Actors url="./acteurs.xml" />
          <Organisations url="./organisation.xml" />
```

```
        <ApplicationParameters>
                <!-- HTTP Config-->
                <!-- Plugin Config -->
        </ApplicationParameters>
        <!-- Config Navigation -->
        <!-- Config Interactions -->
        <!-- Config Scene -->
        <!-- Config Renderer -->
</Application>
```

3.5.1.1 Le serveur HTTP

La configuration du serveur HTTP se situe dans la balise `ApplicationParameters`. Elle se réalise par une balise `HTTP` contenant:

- l'attribut `RessourceDir` définissant le chemin relatif par rapport au fichier .mas du répertoire contenant les ressources statiques du serveur HTTP.
- l'attribut `httpPort` définissant le port du serveur HTTP.

Exemple:

```
<ApplicationParameters>
        <HTTP RessourceDir="HTTPServerBaseDir/" httpPort="9080" />
        <!-- Plugin Config -->
</ApplicationParameters>
```

3.5.1.2 Les déclarations de plugins

La configuration des plugins se situe dans la balise `ApplicationParameters`. Elle se réalise par une balise `Plugins` contenant:

- l'attribut `PluginsDir` définissant le chemin relatif par rapport au fichier .mas du répertoire contenant les plugins.
- de balises `Plugin` contenant un attribut `Name` définissant le nom de la bibliothèque du plugin (par exemple, si le nom est "plugin", l'application cherchera dans le répertoire de plugin un fichier "libplugin.so" sous Linux et "plugin.dll" sous Windows.

Exemple:

```
<ApplicationParameters>
        <!-- HTTP Config-->
        <Plugins PluginsDir="./Dev/">
                <Plugin Name="monPlugin"/>
        </Plugins>
</ApplicationParameters>
```

3.5.2 Enrichissement de péripheriques

Dans Mascaret, un périphérique est un dispositif d'interaction homme machine composé de boutons et d'axes. Un bouton représente une valeur booléenne (le bouton est pressé ou non) et un axe représente une valeur réelle (stick analogique, capteur de position ou autres). Mascaret permet de definir des péripheriques virtuels composé d'axes ou d'enrichir des périphériques existants avec des axes. Ces axes peuvent effectivement être vu comme plusieurs boutons. Ces ajouts pourront ensuite être utiliser dans la configuration de la navigation. Cette configuration se réalise grâce à une balise `Peripherics` contenant des balises `Peripheric`. Ces balises `Peripheric` contiennent:

- l'attribut `name` définissant le nom du nouveau périphérique ou du périphérique à enrichir.
- de balises `ButtonAxis` permettant de définir des axes virtuels

Ces balises `ButtonAxis` contiennent:

- l'attribut `name` définissant le nom de l'axe.
- de balises `Button` définissant les boutons réels à utiliser pour cet axe.

Ces balises `ButtonAxis` contiennent:

- l'attribut `peripheric` définissant le péripherique contenant le bouton.
- l'attribut `button` définissant le nom du bouton à utiliser.
- l'attribut `type` définissant le type d'action réalisé par le bouton. Deux types sont possibles: increase-value ou set-value.
- l'attribut `pressed` définissant si la modification de la valeur de l'axe est réalisée lorsque le bouton est pressé ou lorsqu'il est relaché.
- l'attribut `value` définissant la valeur à utiliser (soit pour augmenter la valeur de l'axe, soit pour la définir selon le `type`). Cette valeur peut être négative.

Exemple (cette configuration permet de définir des axes haut-bas et gauche droite sur le clavier):

```
<Peripherics>
         <Peripheric name="keyboard">
                  <ButtonAxis name="updown">
                           <Button peripheric="keyboard"
                                    button="Up"
                                    pressed="true"
                                    type="set-value"
                                    value="1"/>
                           <Button peripheric="keyboard"
                                    button="Down"
                                    pressed="true"
                                    type="set-value"
                                    value="-1"/>
                           <Button peripheric="keyboard"
                                    button="Up"
                                    pressed="false"
                                    type="set-value"
                                    value="0"/>
                           <Button peripheric="keyboard"
                                    button="Down"
                                    pressed="false"
                                    type="set-value"
                                    value="0"/>
                           <Button peripheric="keyboard"
                                    button="+"
                                    pressed="true"
                                    type="increase-value"
                                    value="1"/>
                           <Button peripheric="keyboard"
                                    button="-"
                                    pressed="true"
                                    type="increase-value"
                                    value="-1"/>
```

```
                    </ButtonAxis>
                    <ButtonAxis name="leftright">
                              <Button peripheric="keyboard"
                                        button="Left"
                                        pressed="true"
                                        type="set-value"
                                        value="1"/>
                              <Button peripheric="keyboard"
                                        button="Right"
                                        pressed="true"
                                        type="set-value"
                                        value="-1"/>
                              <Button peripheric="keyboard"
                                        button="Left"
                                        pressed="false"
                                        type="set-value"
                                        value="0"/>
                              <Button peripheric="keyboard"
                                        button="Right"
                                        pressed="false"
                                        type="set-value"
                                        value="0"/>
                    </ButtonAxis>
          </Peripheric>
</Peripherics>
```

3.5.3 Les interactions

Les interactions permettent d'interagir avec les entités de l'environnement. Elles se déclare grâce à la balise `Interactions`. Les interactions possible peuvent être des types suivants:

- SendSignal: envoie d'un signal à une ou plusieurs entités
- CallOperation: appel une méthode d'une entité
- ChangeViewPoint: change le point de vue
- PlayAnimation: lance une animation
- StopAnimation: arrête une animation

La liste de ces types peut être amené à être enrichit. En effet, il devrait être possible à terme de réaliser n'importe quel `BasicAction` définit dans Mascaret (voir documentation Mascaret). Pour chacun de ces type, la configuration est réalisé par une balise du nom du type possédant les attributs suivants:

- name: définit le nom du signal à envoyé, de l'opération à lancer, du point de vue à définir, de l'animation à jouer ou arrêter.
- target: la cible de l'action. Peut être `broadcast` pour toutes les entités, `designated` pour l'entité désigné par le curseur ou le nom d'une entité.
- peripheric: le nom du périphérique contenant le bouton à utiliser
- button: le nom du bouton à utiliser
- cursor: le curseur permettant la designation (si target="designated"). La valeur par défaut est "system" désignant le curseur souris du système.
- pressed: si l'action doit être lancé lorsque le bouton est appuyé ou relaché (true ou false).

- maxdist: la distance maximale pour laquel l'action est possible (valeur réel positive). La valeur par défaut est infini.
- classifier (CallOperation uniquement): le nom de la classe des entités sur lequels l'opération doit être lancée.

Exemple:

```
<Interactions>
          <SendSignal name="Use" peripheric="mouse" button="button1" pressed="true" target="designated" maxdist
          <CallOperation classifier="Gaspar::Agent::Personnel" name="parler" peripheric="mouse" button="button1
          <ChangeViewPoint name="nomDuViewpoint" direction="" peripheric="keyboard" button="+" pressed="true" t
          <PlayAnimation name="walk" peripheric="mouse" button="button2" pressed="true" target="designated" max
          <StopAnimation name="walk" peripheric="mouse" button="button3" pressed="true" target="designated" max
</Interactions>
```

3.5.4 La navigation

La navigation représente la manière de se déplacer dans l'environnement. Elle est définit par la balise `Navigation` contenant:

- mode: définit le mode de navigation (`free` ou `human`). Le mode `human` permet de ne pas tenir compte de l'angle d'élévation pour les translations (on ne monte pas dans les cieux lorsque l'on avance avec la tête levée)
- la balise `TX`: la translation en x
- la balise `TY`: la translation en y
- la balise `TZ`: la translation en z
- la balise `Roll`: la rotation sur l'axe des x
- la balise `Pitch`: la rotation sur l'axe des y
- la balise `Yaw`: la rotation sur l'axe des z

Les axes de la navigation sont définit par les attributs `peripheric`, `axis` et `speed` représentant respectivement le périphérique comportant l'axe, le nom de l'axe et la vitesse en fonction de la valeur de l'axe.
Exemple:

```
<Navigation mode="human">
          <TX peripheric="spaceMouse" axis="tz" speed="0.01"/>
          <TY peripheric="spaceMouse" axis="tx" speed="-0.01"/>

          <TX peripheric="keyboard" axis="updown" speed="1.5"/>
          <TY peripheric="keyboard" axis="leftright" speed="1.5"/>
          <Yaw peripheric="spaceMouse" axis="ry" speed="0.005"/>
          <Pitch peripheric="spaceMouse" axis="rx" speed="-0.005"/>
          <Yaw peripheric="mouse" axis="x" speed="0.05"/>
          <Pitch peripheric="mouse" axis="y" speed="-0.05"/>
</Navigation>
```

3.5.5 La scene

Exemple:

```
<Scene name="Vue 3D">
          <Decors url="VRMLS/Environement/sceneJourShader.wrl">
                    <Position x="150.0" y="0.0" z="-500"/>
                    <Orientation roll="0.0" pitch="0.0" yaw="3.14"/>
```

```
                    <Light directional="true">
                              <Position x="0" y="-350.0" z="226.0"/>
                              <Orientation roll="0.0" pitch="+0.5" yaw="0.0"/>
                              <Color r="1" g="1" b="1"/>
                    </Light>
                    <Light directional="false">
                              <Position x="0" y="350.0" z="50.0"/>
                              <Orientation roll="0.0" pitch="+0.5" yaw="3.14"/>
                              <Color r="0.2" g="0.2" b="0.2"/>
                    </Light>
                    <Light directional="true" layer="back">
                              <Position x="0" y="-350.0" z="226.0"/>
                              <Orientation roll="0.0" pitch="+0.5" yaw="0.0"/>
                              <Color r="1" g="1" b="1"/>
                    </Light>
                    <Fog>
                              <Properties degree="1" density="2.5"/>
                              <Color r="0.8" g="0.8" b="0.8"/>
                    </Fog>
          </Decors>
</Scene>
```

3.5.6 La fenêtre de rendu

Exemple:

```
<Renderer name="Viewer principal" near="0.4" far="2400">
          <Scene name = "Vue 3D" viewpoint="006" />
          <Window x="0" y="0" width="1919" height="1200" capture-mouse="true"/>
</Renderer>
```

4 Annexe 1: Exemple de fichier environnement complet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Area name="ProgrammateurArea">
        <Viewpoint name="Camera">
                <Position x="0" y="0.17" z="0.8" />
                <Orientation roll="-0.979" pitch="1.57" yaw="0.584" />
        </Viewpoint>
        <Entity name="Prise" class="Programmateur">
           <Shape url="VRMLS/Prise.wrl" movable="false"/>       <!-- La prise sans les bout

           <Entity name="p01" class ="Programme">
              <Attribute name="numero" value="1" />
              <Attribute name="on" value="true" />
           </Entity>
           <Entity name="p02" class ="Programme">
              <Attribute name="numero" value="1" />
              <Attribute name="on" value="false" />
           </Entity>
           <Entity name="p03" class ="Programme">
              <Attribute name="numero" value="2" />
              <Attribute name="on" value="true" />
           </Entity>
           <Entity name="p04" class ="Programme">
              <Attribute name="numero" value="2" />
              <Attribute name="on" value="false" />
           </Entity>
           <Entity name="p05" class ="Programme">
              <Attribute name="numero" value="3" />
              <Attribute name="on" value="true" />
           </Entity>
           <Entity name="p06" class ="Programme">
              <Attribute name="numero" value="3" />
              <Attribute name="on" value="false" />
           </Entity>
           <Entity name="p07" class ="Programme">
              <Attribute name="numero" value="4" />
              <Attribute name="on" value="true" />
           </Entity>
           <Entity name="p08" class ="Programme">
              <Attribute name="numero" value="4" />
              <Attribute name="on" value="false" />
           </Entity>
           <Entity name="p09" class ="Programme">
              <Attribute name="numero" value="5" />
              <Attribute name="on" value="true" />
           </Entity>
```

```
   <Entity name="p10" class ="Programme">
      <Attribute name="numero" value="5" />
      <Attribute name="on" value="false" />
   </Entity>
   <Entity name="p11" class ="Programme">
      <Attribute name="numero" value="6" />
      <Attribute name="on" value="true" />
   </Entity>
   <Entity name="p12" class ="Programme">
      <Attribute name="numero" value="6" />
      <Attribute name="on" value="false" />
   </Entity>
   <Entity name="p13" class ="Programme">
      <Attribute name="numero" value="7" />
      <Attribute name="on" value="true" />
   </Entity>
   <Entity name="p14" class ="Programme">
      <Attribute name="numero" value="7" />
      <Attribute name="on" value="false" />
   </Entity>
        <Attribute name = "programmes" value = "p01" />
        <Attribute name = "programmes" value = "p02" />
        <Attribute name = "programmes" value = "p03" />
        <Attribute name = "programmes" value = "p04" />
        <Attribute name = "programmes" value = "p05" />
        <Attribute name = "programmes" value = "p06" />
        <Attribute name = "programmes" value = "p07" />
        <Attribute name = "programmes" value = "p08" />
        <Attribute name = "programmes" value = "p09" />
        <Attribute name = "programmes" value = "p10" />
        <Attribute name = "programmes" value = "p11" />
        <Attribute name = "programmes" value = "p12" />
        <Attribute name = "programmes" value = "p13" />
        <Attribute name = "programmes" value = "p14" />
        <Attribute name = "programmeCourant" value = "p01" />
</Entity>
<!-- bouton week-->
<Entity name="BoutonWeek" class="BoutonWeek">
        <Position x="-0.0995316527544" y="0.337862770026" z="0.196554875766"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Relation name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonHour" class = "BoutonHeure">
        <Position x="-0.0349308998297" y="0.337862770026" z="0.196554875766"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
```

```
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonMin" class = "BoutonMinute">
        <Position x="0.0332955769951" y="0.337862770026" z="0.196554875766"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonRstRcl" class = "BoutonRST">
        <Position x="0.105317320797" y="0.337862770026" z="0.196554875766"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonClock" class = "BoutonHorloge">
        <Position x="-0.0995316527544" y="0.289167998369" z="0.2024237598"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonProg" class = "BoutonProg">
        <Position x="-0.0349308998297" y="0.289167998369" z="0.2024237598"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonOnAutoOff" class = "BoutonMode">
        <Position x="0.0332955769951" y="0.289167998369" z="0.2024237598"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Bouton.wrl" movable="false"/>
        <Attribute name = "programmateur" value = "Prise" />
</Entity>
<Entity name="BoutonReset" class = "Bouton">
        <Position x="0.105317320797" y="0.289167998369" z="0.205"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/BoutonReset.wrl" movable="false"/>
</Entity>
<Entity name="Ecran" class = "Ecran">
        <Position x="0" y="0.425429088189" z="0.186069580116"/>
        <Orientation roll="-0.0970311585518" pitch="0" yaw="0"/>
        <Shape url="VRMLS/Ecran.wrl" movable="false"/>

        <Entity name="EcranPM" class = "Afficheur">
                <Position x="-0.1" y="0.01" z="0"/>
                <Orientation roll="0" pitch="0" yaw="0"/>
                <Shape url="VRMLS/EcranPM.wrl" movable="false"/>
         </Entity>
```

```
                <Entity name="EcranNum" class = "Afficheur">
                        <Position x="-0.1" y="-0.01" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranNum.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranWeek" class = "Afficheur">
                        <Position x="0.03" y="0.02" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranWeek.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranHour" class = "Afficheur">
                        <Position x="-0.037" y="0.0" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranHour.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranPoint" class = "Afficheur">
                        <Position x="-0.003" y="0.0" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranPoint.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranMin" class = "Afficheur">
                        <Position x="0.03" y="0.0" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranMin.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranSec" class = "Afficheur">
                        <Position x="0.095" y="0.0" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranSec.wrl" movable="false"/>
                 </Entity>
                <Entity name="EcranState" class = "Afficheur">
                        <Position x="0.01" y="-0.02" z="0"/>
                        <Orientation roll="0" pitch="0" yaw="0"/>
                        <Shape url="VRMLS/EcranState.wrl" movable="false"/>
                 </Entity>
        </Entity>
</Area>
```

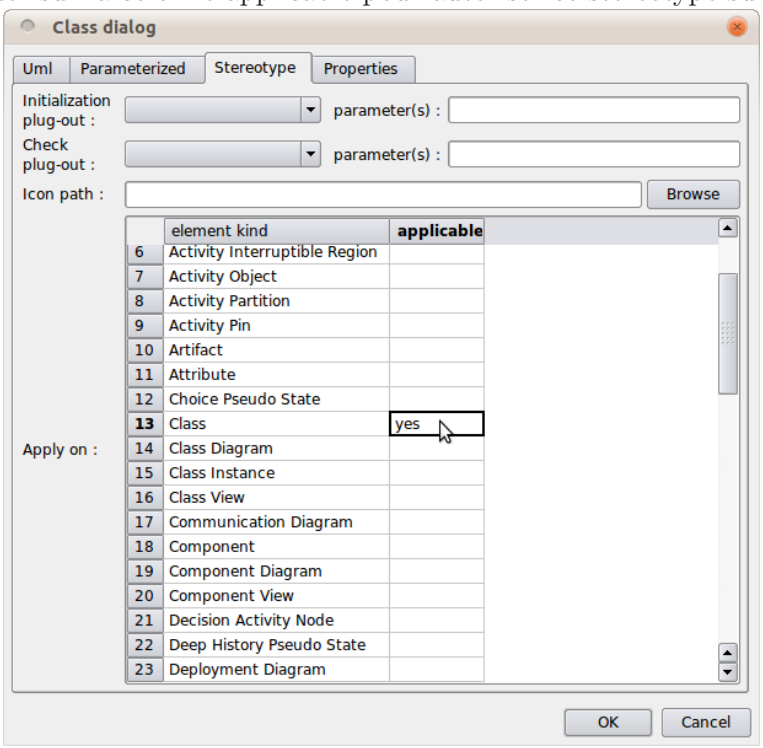
5 Tutorial de création d'une application Mascaret

Dans ce tutorial, nous allons créer un petit environnement Mascaret très simple. Il s'agit simplement d'une horloge type Big Ben sur laquelle il sera possible d'effectuer quelques opérations très simple (ajouter une heure ou une minute).

5.1 Création du modèle

5.1.1 Préparation du projet

1. Lancer BOUML et créer un nouveau projet appelé ClockTower.
2. A la racine du projet, faite bouton droit, New Profile. Appelez le Mascaret.
3. Sur ce profile, faite un click droit, New class/stereotype view. Appelez le Mascaret aussi.
4. Enfin sur ce dernière élément, faites click droit, New stereotype. Appelez le Entity.
5. Double cliquer sur ce stéréotype. Dans l'onglet Stereotype, recherche le `element kind` Class et cliquer sur la colonne applicable pour autoriser ce stéréotype sur les classes.

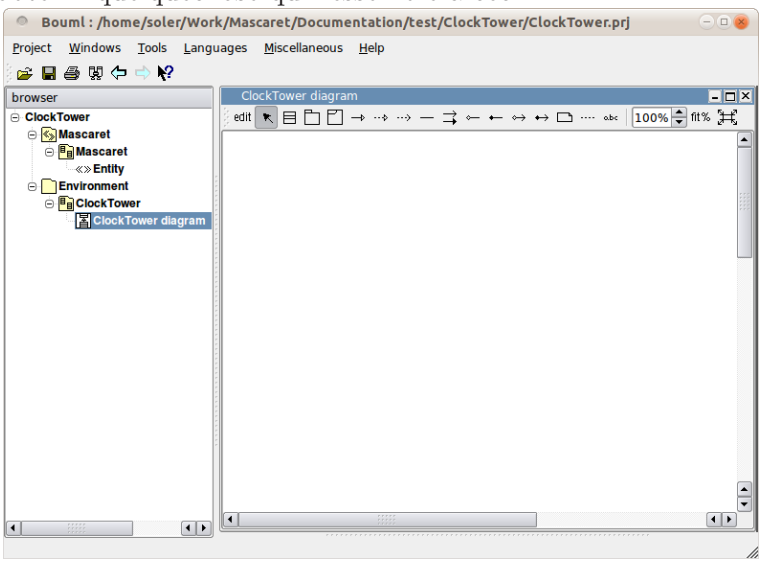


Ici, nous n'aurons besoin que de ce stéréotype. Si vous avez besoin d'un autre stéréotype Mascaret (Role, Team, ..., voir Chapitre 3 [Creation application], page 5)

6. Faites click droit sur la racine du projet et créer un nouveau package. Appelez le Environment.
7. Faites click droit sur ce package et créer un class view. Appelez le ClockTower.
8. Créer y maintenant un class diagram.

9. Double cliquer sur ce dernier

Vous devez obtenir quelquechose qui ressemble à ceci:



5.1.2 Création du modèle statique

Nous allons maintenant décrire ce qui va constituer notre environnement. A savoir, une tour avec une horloge.

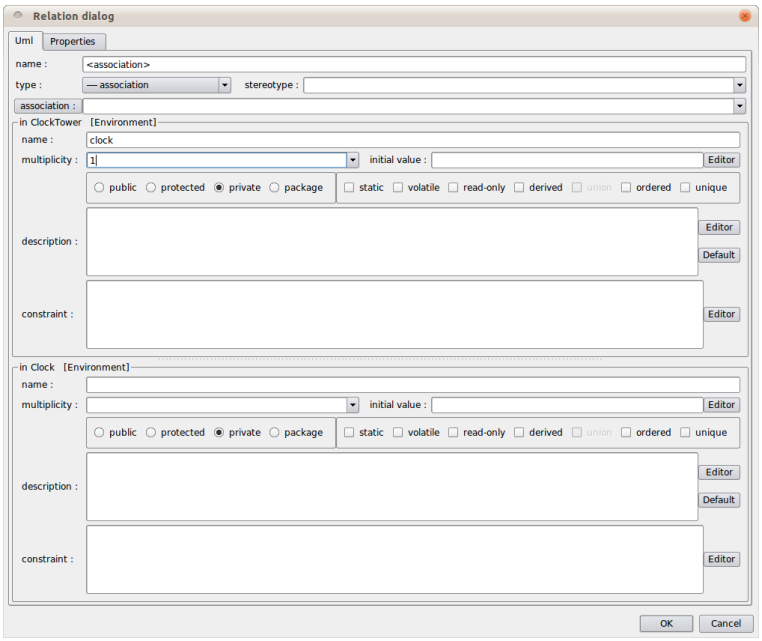
10. Ajouter une classe sur le diagramme (l'icône ) et appelez la `ClockTower`

11. Créer en une 2ème appelée `Clock`

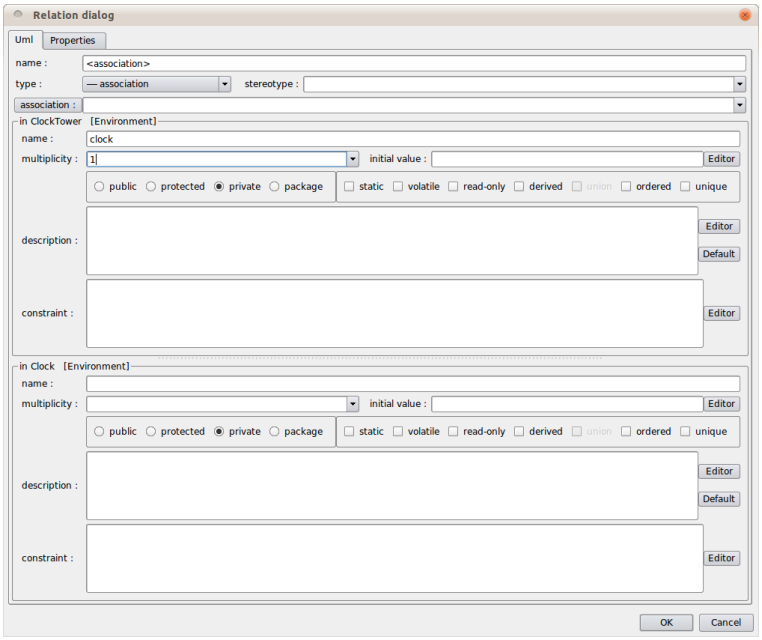
12. Créer en une 3ème appelée `Hand` (une aiguille)

13. Une `ClockTower` contient une `Clock`. Pour exprimer ceci, faite une association entre `ClockTower` et `Clock` (icône)

14. Double clicker sur ce lien et donnez lui le nom clock avec une multiplicité de 1.



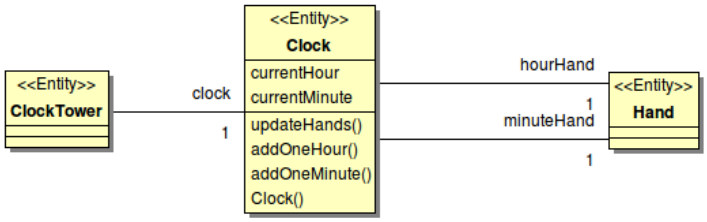
15. Une `Clock` contient 2 `Hand`. Pour exprimer ceci, faite 2 associations entre `Clock` et `Hand`. Nommez la première hourHand (la petite aiguille) et l'autre minuteHand (la grande aiguille).
16. Dans l'arborescence de gauche, faites un bouton droit sur la classe `Clock` pour ajouter un attribut. Appelez le `currentHour`.
17. Double clickez sur cette attribut. Donnez lui le type `int` et la valeur par défaut `"0"`



18. De même, créer un attribut `currentMinute`.
19. Créer une méthode dans la classe `Clock` en faisant bouton droit, New operation. Appelez la `updateHands` (elle ne prend pas de paramètre, il faut juste renseigner son nom).

20. De même, créez une méthode `addOneHour`, une autre `addOneMinute` et encore une autre `Clock` (un constructeur, non verront après pourquoi).

Vous devez obtenir ceci:



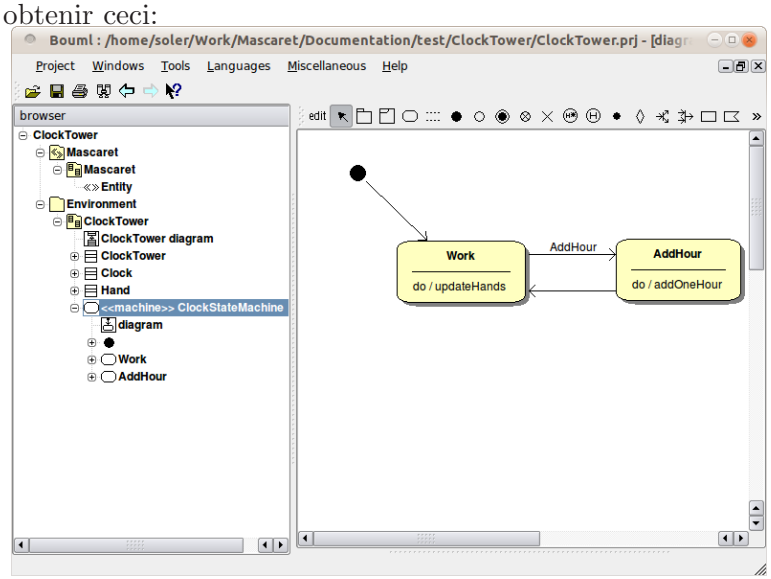
Nous avons donc décrit la partie statique des entités qui vont peupler notre environnement. Nous pourrons donc, dans notre environnement, instancier une (ou plusieurs) `ClockTower` qui contiendra les éléments contenu dans le modèle. Les attributs `currentHour` et `currentMinute` de la classe `Clock` vont permettre de "stocker" l'heure que l'horloge va afficher. La méthode `updateHands` va permettre de mettre à jour la position des aiguilles en fonction de cette heure courante. Les méthode `addOneHour` et `addOneMinute` vont permettre d'augmenter de 1 les valeurs respective des attributs `currentHour` et `currentMinute`.

5.1.3 Création du modèle dynamique

Nous allons maintenant décire le comportement dynamique de cette horloge.

21. Créer une machine à état dans le class view ClockTower (bouton droit, New state machine). Appelez la ClockStateMachine.
22. Double clickez sur cette machine à état et dans le combobox Specification, choisir la méthode `Clock()`. En faite, il s'agit d'associer cette machine à état à la classe `Clock` de manière à ce qu'une instance de cette machine à état soit exécuter après l'instanciation d'une `Clock`. Il est possible d'associer de cette manière plusieurs machine à état à une classe. Il faut toutefois noté qu'il s'agit ici d'un contournement de BOUML qui ne permet pas de déclarer directement une machine à état dans une classe.
23. Sur cette machine à état, créer un diagramme (New state diagram) et double cliquer dessus pour l'ouvrir.
24. Ajouter un état initial au diagramme (icône )
25. Ajouter un état Work et un état AddHour (icône )
26. Ajouter une transition de l'état initial à l'état Work, une de l'état Work à l'état AddHour et une de l'état AddHour à l'état Work.
27. Double clicker sur la transition de l'état Work à l'état AddHour.
28. Dans l'onglet OCL, dans la partie trigger, saisissez le nom de signal `AddHour`.

Vous devez obtenir ceci:



Nous venons de définir ici un comportement très simple de l'horloge. Lorsqu'elle recevra le signal `AddHour`, elle passera de l'état `Work` à l'état `AddHour` et executera la méthode `addOneHour` avant de revenir à l'état Work et de réexecuter la méthode `updateHands`. Nous verrons par la suite comment implémenter ces méthodes et comment envoyé un signal à une instance.

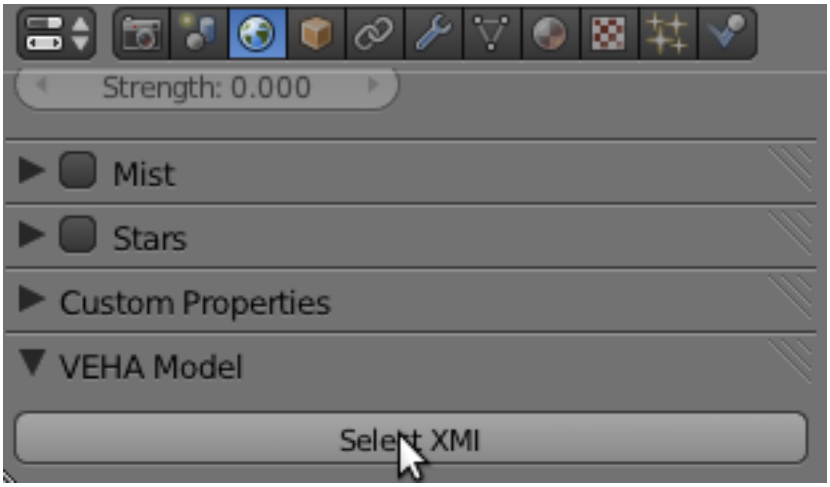
La création du modèle est terminée. Il ne reste plus qu'à exporter le modèle en XMI. Pour cela, faites un click droit à la racine du projet, Tool > Generate XMI 2.1. Choisissez le nom et l'emplacement du fichier à exporter.

5.2 Instanciation de l'environnement

Le but de cette partie est d'instancier un environnement correspodant au modèle. Nous allons pour cela utiliser le logiciel Blender et l'addon Mascaret pour Blender. L'installation de l'addon est décrite dans ce document (voir Chapitre 3 [Creation application], page 5)

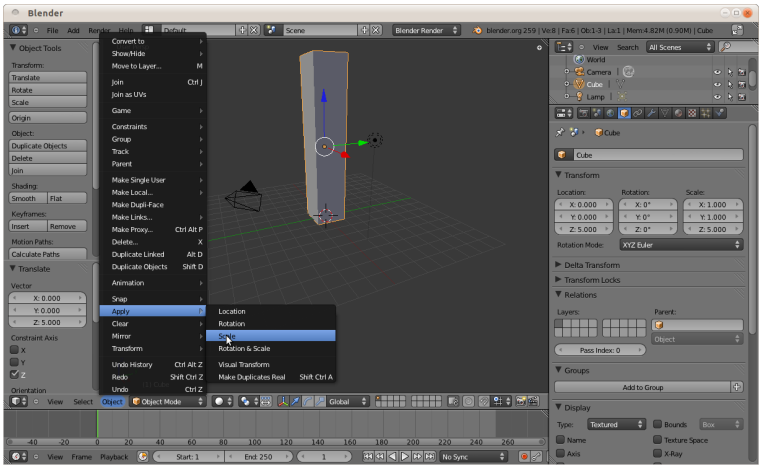
1. Ouvrir Blender

2. Dans la fenêtre de propriétés de Blender, dans l'onglet World, section VEHA Model, appuyez sur le bouton Select XMI et choissez le fichier XMI précédemment exporté.



Le but de se document n'étant pas de former à la modélisation 3D, nous représenterons la tour par un parrallélépipède, l'horloge par un cylindre et les aiguilles par 2 parallélépipèdes.

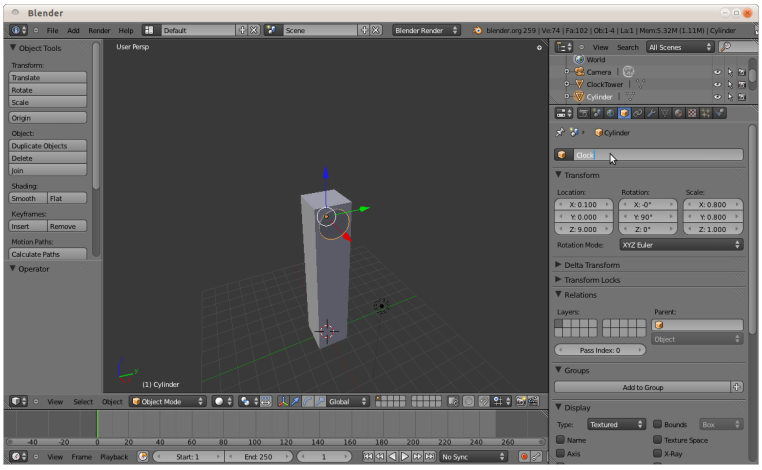
3. Faites un click droit sur le cube déjà présent, appuyez sur s, puis z, puis saisissez [5 enter].
4. Pour faire remonté la tour sur l'axe z de manière à ce que le sol soit le plan z=0, faite g, puis z, puis [5 enter].
5. Nous avons effectué une mise à l'échelle (avec le raccourci s) pour créer cette tour. Mascaret ne gère pas la mise à l'échelle. Il faut donc appliqué cette mise à l'échelle au mesh. Pour cela, dans le menu objet (menu de la vue 3D), faites Apply > Scale.



De manière générale, il faut vérifier qu'aucun objet de votre scène n'a une échelle différente de (1,1,1). On peut voir cette échelle dans les propriétés de l'objet (sur la droite de l'image ci-dessus).

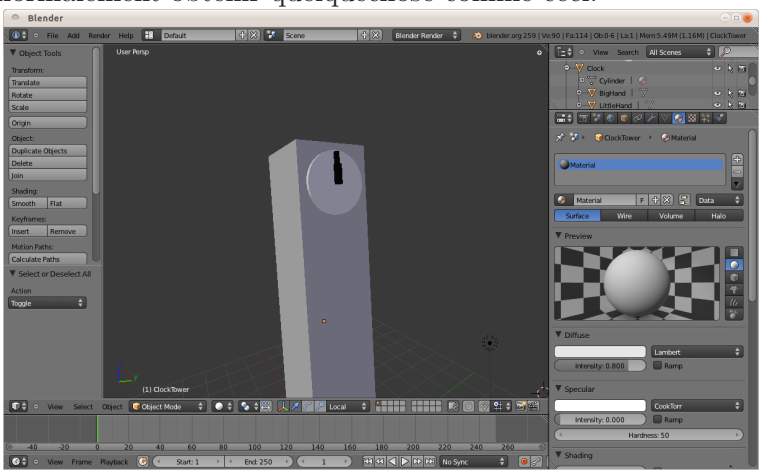
7. Dans les propriétés de l'objet, renommez l'objet en ClockTower (à la place de Cube).
8. Ajouter un cylindre en faisant Add (du menu principal) > Mesh > Cylinder.
9. Le cylindre qui apparait est en partie masqué par la tour. Pour basculer entre la vue scene à la vue contextuelle de la selection, vous pouvez appuyer sur la touche "/" du pavé numérique. Vous pouvez aussi centrer la vue sur la selection avec la touche "." du pavé numérique.
10. Pour orienté correctement le cylindre, faites r, y, et saisissez [90 enter]

11. Le cylindre est un peu trop grand. Ajustez-le en faisant une mise à l'echelle sur l'axe y : s, y, [0.8 enter] puis s, z, [0.8 enter].
12. Appliquez l'échelle au mesh.
13. Faites dépasser légèrement le cylindre de la tour : g, x, [0.1 enter]
14. Monter le cylindre en haut de la tour : g, z, [9 enter].
15. Renommer l'objet en Clock (à la place de Cylinder).



16. Ajouter un cube (Add > Mesh > Cube).
17. Appuyer sur la touche "/" du pavé numérique pour visualisez ce cube seul.
18. Mise à l'echelle sur y et z: s, y, [0.05 enter], s, z, [0.4 enter]
19. Appliquer la mise à l'echelle au mesh
20. Il nous faut maintenant modifier le centre de l'aiguille car dans la simulation, les rotations de l'objet sont exprimées par rapport à lui. Nous voulons que les aiguilles tourne par rapport au centre de l'horloge, c'est à dire le bas de l'aiguille. Pour cela, monter l'aiguille de 0.35 en z (g, z, [0.35 enter]) et faites Object > Apply > Location.
21. Appuyer sur la touche "/" du pavé numérique pour voir toute la scene.
22. Dans la section Relation des propriétés de l'objet, définir le parent de l'aiguille à Clock
23. Avancer l'aiguille: gx 0.1
24. Renommer l'objet en BigHand (à la place de cube).
25. Nous allons maintenant repeté les mêmes étapes pour réaliser la petite aiguille: Ajouter un cube
26. Mise à l'echelle sur y et z: s, x, [0.07 enter], s, z, [0.25 enter]
27. Appliquer la mise à l'echelle au mesh
28. g, z, [0.2 enter] et faites Object > Apply > Location.
29. Définir le parent de l'aiguille à Clock
30. Avancer l'aiguille: gx 0.15
31. Ajouter un materiaux au divers objet afin de changer leur couleurs.

Vous devez normalement obtenir quelquechose comme ceci:



Il ne reste plus à ce stade qu'à definir le type de chaque objet graphique.

32. Selectionner la tour. Dans les propriétés de l'objet, dans la section VEHA Model, définir l'attribut class à `ClockTower`.
33. Selectionner l'horloge. Dans les propriétés de l'objet, dans la section VEHA Model, définir l'attribut class à `Clock`.
34. Selectionner une aiguille. Dans les propriétés de l'objet, dans la section VEHA Model, définir l'attribut class à `Hand` et faites de même avec l'autre.
35. Dans les propriétés de l'objet, dans la section VEHA Model, tout en bas, clicker sur le bouton Update references.
36. Selectionner l'horloge. Dans les propriétés de l'objet, dans la section VEHA Model, définir l'attribut hourHand à `LittleHand`. Définir l'attribut minuteHand à `BigHand`.
37. Selectionner la tour. Dans les propriétés de l'objet, dans la section VEHA Model, définir l'attribut clock à `Clock`.

L'environnement est maintenant fini. Vous pouvez maintenant éventuellement ajouter des décors, texturer les objets, placer et orienté des caméra qui consituront les points de vue de la scene... Tous les objets qui n'ont pas de classe définis seront considérés comme du décors mais seront bien présent dans la simulation.

Pour exporter la scene, faites File > Export > Export VEHA Environment et choisissez le fichier d'export. Des fichiers au format x3d sont généré à coté du fichier d'export. Pour qu'il puissent être lu par AReViMascaret, il faut exécuter la feuille de style X3dToVrml97.xslt fournit avec l'editeur d'environnement Blender. Par exemple, sous linux:

```
for i in $(ls *.x3d); do xsltproc X3dToVrml97.xslt $i>${i%%.*}.wrl ; done
```

5.3 Ecriture du plugin

Le plugin va nous permettre d'implémenter les comportements opaques (les méthodes de la classe `Clock`) et d'initialiser l'heure courante de l'horloge à l'heure du système.

5.3.1 Initialisation de la simulation

Nous allons utilisé un des évènements de Mascaret: La fonction `MASCARET_MODEL_LOADED`. Nous allons récupérer l'heure du système et mettre à jour les attributs `currentHour` et `currentMinute`.

```
#include "VEHA/Behavior/Common/BehaviorExecution.h"
#include "VEHA/Behavior/Common/Behavior.h"
#include "VEHA/Entity/Entity.h"
#include "VEHA/Entity/RotationVector.h"
#include "VEHA/Kernel/LiteralInteger.h"

#include "MascaretApplication.h"

#include <sys/time.h>
using namespace VEHA;


extern "C"
VEHA_PLUGIN void MASCARET_MODEL_LOADED()
{
          // Get system time
          time_t now;
          struct tm* tm;
          now = time(0);
          tm = localtime(&now);

          // Get the clock
          shared_ptr<InstanceSpecification> clock=MascaretApplication::getInstance()->getEnvironment()->getInst
          // Set currentHour and currentMinute properties with system time
          clock->getProperty("currentHour")->addValue(LiteralInteger(tm->tm_hour));
          clock->getProperty("currentMinute")->addValue(LiteralInteger(tm->tm_min));

          // We could launch the updateHands operation but it's not necessary (will be launch by state machine)
          //InstanceSpecification::OperationCallParameters params;
          //clock->executeOperation("updateHands",params);
}
```

5.3.2 Implémentation des comportements opaques

```
class Clock_updateHands : public BehaviorExecution
{
public:
          Clock_updateHands(shared_ptr<Behavior> specif, shared_ptr<InstanceSpecification> host, const Paramete
          : BehaviorExecution(specif, host, p)
          {
                    // Get currentHour and currentMinute slots
                    _currentHour=getHost()->getProperty("currentHour");
                    _currentMinute=getHost()->getProperty("currentMinute");
                    // Get hands from slots
                    _hourHand=shared_dynamic_cast<Entity>((shared_ptr<InstanceSpecification>)*getHost()->getProp
                    _minHand=shared_dynamic_cast<Entity>((shared_ptr<InstanceSpecification>)*getHost()->getPrope
          }
          virtual ~Clock_updateHands()
          {
          }

          double execute()
          {
```

```
                    // Get currentHour and currentMinute values from slots
                    int hour=*_currentHour->getValue() % 12;
                    int min=*_currentMinute->getValue();
                    // Compute hands angles
                    double hourAngle=-hour/12.0*2*M_PI;
                    double minAngle=-min/60.0*2*M_PI;
                    // Set hands angles
                    _hourHand->setLocalRotation(RotationVector(1,0,0,hourAngle));
                    _minHand->setLocalRotation(RotationVector(1,0,0,minAngle));
                    return 0;
          }
          shared_ptr<Slot> _currentHour;
          shared_ptr<Slot> _currentMinute;
          shared_ptr<Entity> _hourHand;
          shared_ptr<Entity> _minHand;
};

extern "C"
VEHA_PLUGIN BehaviorExecution* Clock_updateHands_init(shared_ptr<Behavior> specif, shared_ptr<InstanceSpecifica
{
          return new Clock_updateHands(specif, host,p);
}
```

5.4 Configuration de la simulation

5.4.1 Référencer le modèle, l'environnement et les plugins

```
<?xml version="1.0" encoding="UTF-8"?>
<Application>
          <Model url="ClockTower.xmi" xmi2="true"/>
          <Environment url="ClockTower.xml" />
          <ApplicationParameters>
                    <!-- HTTP Config-->
                    <Plugins PluginsDir="./ClockPlugin">
                              <Plugin Name="ClockPlugin"/>
                              <Plugin Name="SpaceMouseMascaret"/>
                    </Plugins>
          </ApplicationParameters>
</Application>
```

5.4.2 Définir les paramètres de rendu

```
<Scene name = "scene">
          <Decors>
                    <Light name="Lamp" directional="false">
                              <Position x="4.076245307922363" y="1.0054539442062378" z="5.903861999511719"/>
                              <Orientation roll="0.6503279805183411" pitch="0.055217113345861435" yaw="1.8663908
                    </Light>
          </Decors>
</Scene>
<Renderer near="0.1" far="1000" fieldOfView="45">
          <Window x="0" y="0" width="700" height="1000" capture-mouse="true" />
          <!--Stereo /-->
          <Scene name="scene" viewpoint="Camera" />
</Renderer>
```

Nous pouvons maintenant lancer la simulation:
`areviMascaret ClockTower.mas`

5.4.3 Ajouter de l'interaction

```
<Interactions>
         <SendSignal name="AddHour" peripheric="mouse" button="button1" pressed="true" target="designated"/>
         <CallOperation classifier="ClockTower::ClockTower::Environment::Clock" name="addOneMinute" peripheric
         <CallOperation classifier="ClockTower::ClockTower::Environment::Clock" name="updateHands" peripheric=
</Interactions>
```

5.4.4 Ajouter la navigation

```
<Navigation type="camera" mode="human">
         <TX peripheric="spaceMouse" axis="tz" speed="0.01"/>
         <TY peripheric="spaceMouse" axis="tx" speed="-0.01"/>
         <!--TZ peripheric="spaceMouse" axis="ty" speed="0.01"/-->
         <Yaw peripheric="spaceMouse" axis="ry" speed="0.005"/>
         <Pitch peripheric="spaceMouse" axis="rx" speed="-0.005"/>
</Navigation>
<Peripherics>
         <Peripheric name="keyboard">
                  <ButtonAxis name="updown">
                           <Button peripheric="keyboard" button="Up" pressed="true" type="set-value" value="1
```

```
                        <Button peripheric="keyboard" button="Down" pressed="true" type="set-value" value=
                        <Button peripheric="keyboard" button="Up" pressed="false" type="set-value" value="
                        <Button peripheric="keyboard" button="Down" pressed="false" type="set-value" value
                        <Button peripheric="keyboard" button="+" pressed="true" type="increase-value" valu
                        <Button peripheric="keyboard" button="-" pressed="true" type="increase-value" valu
                </ButtonAxis>
                <ButtonAxis name="leftright">
                        <Button peripheric="keyboard" button="Left" pressed="true" type="set-value" value=
                        <Button peripheric="keyboard" button="Right" pressed="true" type="set-value" value
                        <Button peripheric="keyboard" button="Left" pressed="false" type="set-value" value
                        <Button peripheric="keyboard" button="Right" pressed="false" type="set-value" valu
                </ButtonAxis>
        </Peripheric>
</Peripherics>
<Navigation type="camera" mode="human">
        <TX peripheric="spaceMouse" axis="tz" speed="0.01"/>
        <TY peripheric="spaceMouse" axis="tx" speed="-0.01"/>
        <TX peripheric="keyboard" axis="updown" speed="1.5"/>
        <TY peripheric="keyboard" axis="leftright" speed="1.5"/>
        <Yaw peripheric="spaceMouse" axis="ry" speed="0.005"/>
        <Pitch peripheric="spaceMouse" axis="rx" speed="-0.005"/>
        <Yaw peripheric="mouse" axis="x" speed="0.05"/>
        <Pitch peripheric="mouse" axis="y" speed="-0.05"/>
</Navigation>
```

Index