

GenCSVcan, un composant SIG générique pour RECORD

E. Casellas, P. Chabrier, H. Raynal. Septembre 2012
vle ≥ 1.1.0-dev-abe6a06

Spécifications fonctionnelles attendues:

On veut dans le cadre du projet RECORD mettre à la disposition des modélisateurs un modèle générique (c'est à dire facilement réutilisable, et ce dans une grande variété de contextes), qui puisse à partir d'un fichier CSV de description d'un parcellaire (contenant l'identification des parcelles et des attributs associés), instancier la structure d'un modèle VLE correspondant.

On a décidé de se restreindre à un cas particulier dans lequel on a :

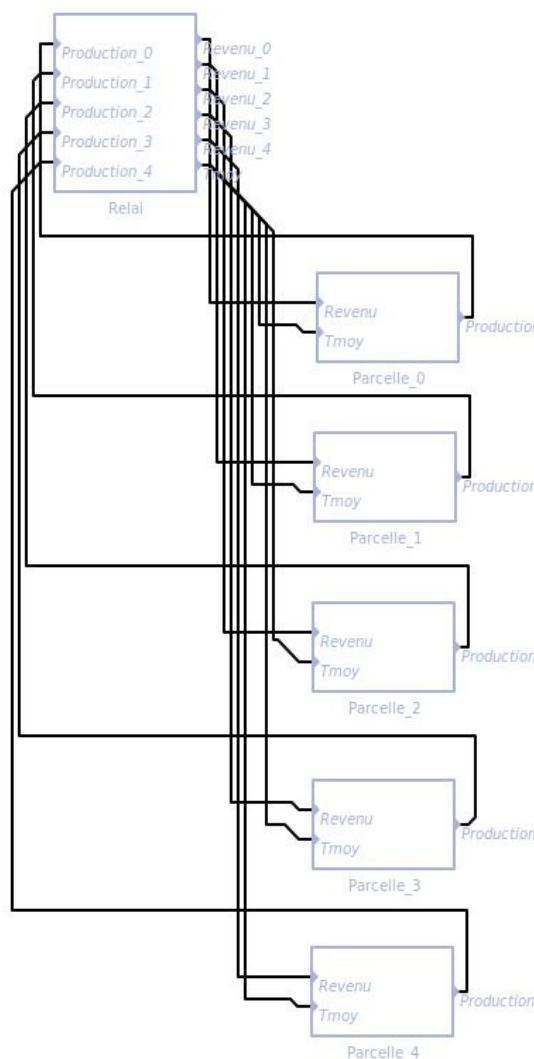
- Un fichier CSV contenant n parcelles et des attributs associés.

- Une structure fixe mise en place à l'initialisation

- Une unique classe de modèle *cParcelle* dupliquée pour chaque parcelle (dynamiques/ports/observables identiques mais avec possibilité de paramétrage spécifique par parcelle selon les informations contenues dans le fichier SIG). n parcelles ayant k ports d'entrée ($k = l + m$ avec l le nombre d'entrées communes à toutes les parcelles et m le nombre d'entrées spécifiques à chaque parcelle) et j ports de sortie ($j = n + o$ avec n le nombre de sorties communes à toutes les parcelles et o le nombre de sorties spécifiques à chaque parcelle).

- Aucun liens direct entre les parcelles.

- Un modèle *Relai* qui est lui connecté à toutes les parcelles (possibilité de paramétrage en fonction des informations contenues dans le fichier CSV – autre que juste le nombre de parcelles). $n*m + l$ ports de sorties et $n*o + n$ ports d'entrées.



*Fig.1: exemple du schéma de connections entre le modèle **Relai** et n modèle **Parcelle** ($n=5$) tel que généré par GenCSVcan. (modèles parcelles avec $l=1$ entrée commune et $m=1$ entrée spécifique, $n=0$ sorties commune et $o=1$ sorties spécifiques)*

Spécification non-fonctionnelle:

GENeric CSV Connecting All to oNe (GenCSVcan)
vle 1.1.0-dev-abe6a06

Extension VLE:

vle::devs::Executive

Paramètres du modèle GenCSVcan:

Nom du port	Data Type	Default value	Description
CsvFileName	string	-	Nom du fichier CSV à lire.
ParcelleClassName	string	cParcelle	Nom de la classe du vpz à utiliser pour les modèles Parcelle.
ParcellePrefix	string	Parcelle	Préfixe utilisé pour les instances des différentes Parcelles (voir nomenclature).
RelaiName	string	Relai	Nom du modèle Relai.
ConditionSeparator	string	;	Séparateur des conditions en listes
CSVSeparator	string	;	Séparateur utilisé pour les différents elements du fichier .csv
R_inputPortsInd	string (*)	-	Noms des ports d'entrée du modèle Relai. (* liste avec des espaces comme séparateur).
R_inputPortsCom	string (*)	-	Noms des ports d'entrée du modèle Relai. (* liste avec ConditionSeparator comme séparateur).
R_outputPortsInd	string (*)	-	Noms des ports de sortie du modèle Relai qui sont spécifiques à chaque Parcelle (voir nomenclature). (* liste avec ConditionSeparator comme séparateur).
R_outputPortsCom	string (*)	-	Noms des ports de sortie du modèle Relai qui sont communes à toutes les parcelles (voir nomenclature). (* liste avec ConditionSeparator comme séparateur).
R_Condition	string	-	Nom de la condition du modèle couplé Relai qui contient le paramètre n du nombre de parcelles.
P_Condition_id	string	-	Nom de la condition des modèles parcelle qui contient le paramètre id identifiant de la parcelle
P_Conditions	string (*)	-	Nom des conditions qui contiennent les paramètres spécifiques à chaque parcelle. (* liste avec ConditionSeparator comme séparateur)
P_ParamPorts	string (*)	-	Noms des ports la condition P_Conditions des paramètres spécifiques à chaque parcelle. (* liste avec ConditionSeparator comme séparateur).
P_ParamPortsType	string (*)	-	Types des ports la condition P_Condition des paramètres spécifiques à chaque parcelle. (* liste avec ConditionSeparator comme séparateur). NB: seul les types {"double"; "integer"; "string"} sont pris en charge.
P_ParamPortsDBName	string (*)	-	Noms des champs de la BDD du fichier SIG contenant les valeurs des paramètres spécifiques à chaque parcelle. (* liste avec ConditionSeparator comme séparateur).
CsvFileDecimalLocale	string	en_US	Nom de l'option "locale" a utiliser pour le séparateur

			décimale (paramètre optionnel, utilise la valeur par défaut "en_US" ~ séparateur point. utiliser "fr_FR" pour des virgules).
DbgLog	double	0	Niveau de verbosité. >=1 produit un dump des modifications du vpz faites par ce modèle

Note : les paramètres sur fond gris sont facultatifs, ceux en gris foncé doivent avoir le même nombre d'éléments.

Workflow:

-État initial:

Doivent être présent dans le vpz:

- Le modèle GenCSVcan correctement configuré (voir liste des paramètres ci-dessus).
- Un modèle **Relai** sans ses ports d'entrée/sortie (doit être capable de gérer la génération dynamique des variables d'entrées / sorties spécifiques à chaque Parcelle).
- La classe **cParcelle** correctement configurée (possédant elle déjà ses ports d'entrée/sortie et ses conditions/observations/dynamique attachées).

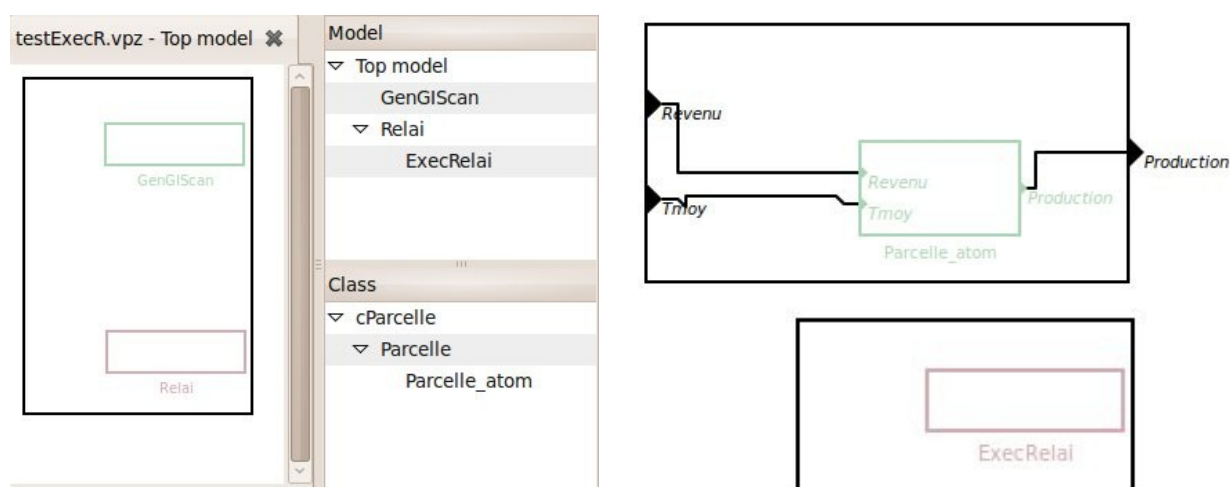


Fig.2: état initial du vpz utilisant GenCSVcan.

- A gauche le modèle complet contenant le modèle GenCSVcan et le modèle couplé **Relai**.
- En haut à droite un exemple de classe **cParcelle** du modèle des parcelles (voir code source `src/test/testParcelle.cpp` du pkgs). Une classe est un graph de modèle DEVS (avec toutes les informations nécessaires : dynamiques, conditions, observations)
- En bas à droite un exemple de modèle couplé **Relai**, contenant un modèle executif (voir code source `src/test/ExecRelai.cpp` du pkgs), qui va instancier les modèles atomiques du modèle **Relai** (voir code source `src/test/testRelai.cpp` du pkgs)

-Constructeur de GenCSVcan:

Le modèle récupère les différentes valeurs de paramètres du vpz (voir tableau des paramètres ci-dessus), et en vérifie (partiellement) la validité : en particulier **P_Conditions**, **P_ParamPorts**, **P_ParamPortsType**, et **P_ParamPortsDBName** doivent contenir le même nombre d'éléments.

-init() de GenCSVcan:

1- Lecture du fichier CSV afin de récupérer: le nombre de parcelles à simuler (on prend toutes celles définies dans le fichier `CsvFileName`), ainsi que les différents attributs de chaque parcelles (les noms de ces attributs sont à définir via le paramètre **P_ParamPortsDBName**).

2- Mise à jour du paramètre **n** du modèle couplé **Relai** en fonction du nombre de parcelles trouvées dans le fichier CSV. (c'est à partir de ce paramètre **n** que le modèle **Relai** doit être capable de gérer de façon autonome ses connections internes ainsi que la déclaration de ses Var).

3- Créations des ports d'entrée / sortie du modèle **Relai**:

R_inputPortsInd (n ports d'entrée pour chaque élément de cette liste de ports, voir nomenclature)

R_inputPortsCom (un unique port d'entrée pour chaque élément de cette liste de ports)

R_outputPortsInd (n ports de sortie pour chaque élément de cette liste de ports, voir nomenclature)

R_outputPortsCom (un unique port de sortie pour chaque élément de cette liste de ports)

4- Modification des conditions spécifiques de chaque parcelle (peuvent être soit des paramètres des modèles **Parcelle** ou bien du modèle **Relai**):

On utilise les valeurs récupérées dans le fichier CSV (via **P_ParamPortsDBName**), pour modifier les ports **P_ParamPorts** des conditions **P_Conditions** (et de type **P_ParamPortsType**)

5- Ajout des **n** modèles **Parcelle**:

n modèles (nommés **ParcellePrefix_X** avec X un index entier allant de 0 à n-1) sont créés à partir de la classe définies par le paramètre **ParcelleClassName**.

6- Connexions des entrées / sorties du modèle **Relai** et les **n** modèles **Parcelle** :

En utilisant nos règles de nomenclature (préfixe des noms de ports identiques), tous les ports sont connectés entre eux.

-Exemple de constructeur d'un modèle Executive contenu dans le modèle couplé

Relai:

Le modèle récupère les différentes valeurs de paramètres du vpz (typiquement il aura besoin de partager les paramètres de GenCSVcan suivant: **R_inputPortsInd**, **R_inputPortsCom**, **R_outputPortsInd**, **R_outputPortsCom**. Un autre paramètre important est le **n** correspondant aux nombre de parcelles à simuler).

-Exemple de init() d'un modèle Executive contenu dans le modèle couplé Relai:

1- Modification de la condition contenant la déclaration des Var du modèle Relai (contenant les différentes variables d'état de modèles DifferenceEquation). On déclare ici les n répétitions des sorties définies par le paramètre **R_outputPortsInd**, ainsi que les sorties définies par **R_outputPortsCom**

2- Ajout d'un modèle atomique **Relai_atom** avec ses différents ports d'entrée/sortie.

3- Ajout des différentes sorties du modèle aux observations.

4- Connexions des ports d'entrée/sortie du modèle atomique **Relai_atom** avec les ports du modèle couplé **Relai**

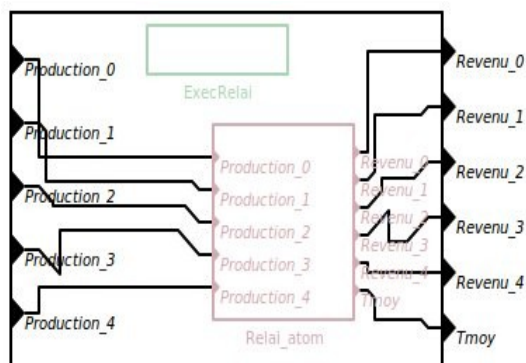


Fig.3: exemple de l'état final du modèle couplé Relai après action du modèle Executive qu'il contient.

Contraintes sur le modèle Relai:

-Doit être capable de créer les liens entre ses ports d'entrée générés par GenCSVcan (voir nomenclature et les paramètres **R_inputPortsInd**, **R_inputPortsCom**) et les ports d'entrées des modèles atomiques définis dans le modèle couplé **Relai**. (à partir d'un paramètre **n** donnant le nombre de parcelles)

-Doit être capable de créer les liens entre ses ports de sortie générés par GenCSVcan (voir nomenclature et les paramètres **R_outputPortsInd**, **R_outputPortsCom**) et les ports de sortie des modèles atomiques définis dans le modèle couplé **Relai**. (à partir d'un paramètre **n** donnant le nombre de parcelles)

(note : les contraintes énoncées ci-dessus ne s'appliquent pas dans le cas où on n'utilise que des ports d'entrées/sorties communs et pas de port spécifique)

Contraintes sur la classe de modèle *cParcelle*:

- Nomenclature pour les ports entrée/sortie doit correspondre à celle du modèle couplé *Relai*
- Présence des Conditions/Observation/Dynamiques (dont déclaration des Var et des paramètres spécifiques de chaque parcelle).

Nomenclature à respecter:

Pour pouvoir connecter les variables d'entrée / sortie du modèle *Relai* et celles des modèles *Parcelle*, il faut adopter une convention de nomenclature: les noms doivent être du type VarX_1, VarX_2,..., VarX_n avec un préfixe commun (ici VarX_1 est la sortie VarX de la classe Relai qui ira vers la parcelle 1) et une numérotation en fonction de la parcelle vers qui le lien sera fait (ceci n'est pas nécessaire pour les variables de sortie du modèle Relai qui sont identiques entre les parcelle). De façon similaire il faut que les entrées du modèle Relai suivent le même type de nomenclature InputX_1, InputX_2,..., InputX_n (où InputX_1 est l'entrée InputX qui viens de la parcelle 1).

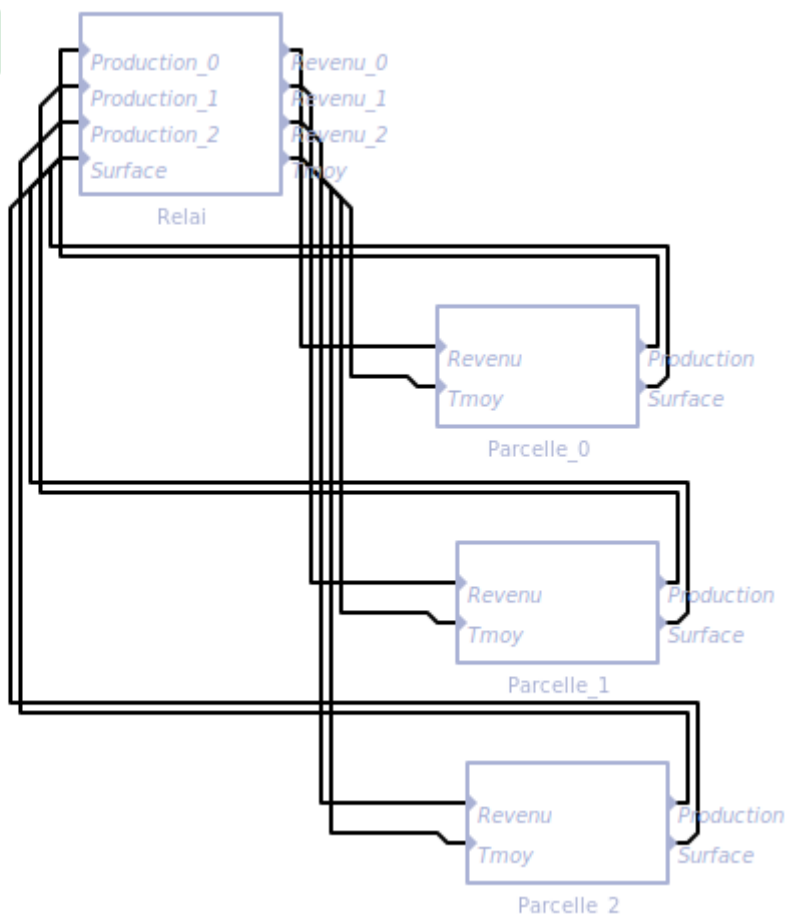
Les noms des ports entre le modèle Relai et les modèles Parcelle doivent partager les même préfixes (on connecte les entrées et sorties qui ont le même préfixe)

Annexe : Cas d'utilisation, graph générés par GenCSVcan

(fichiers dump "output/output_GenCSVcan.vpz")

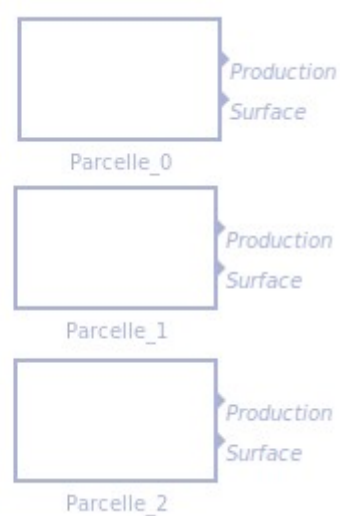
GenCSVcan Full.vpz :

R_inputPortsInd > 0
R_inputPortsCom > 0
R_outputPortsInd > 0
R_outputPortsCom > 0



GenCSVcan NoRelai.vpz :

R_inputPortsInd = 0
R_inputPortsCom = 0
R_outputPortsInd = 0
R_outputPortsCom = 0



GenCSVcan PR XX.vpz :

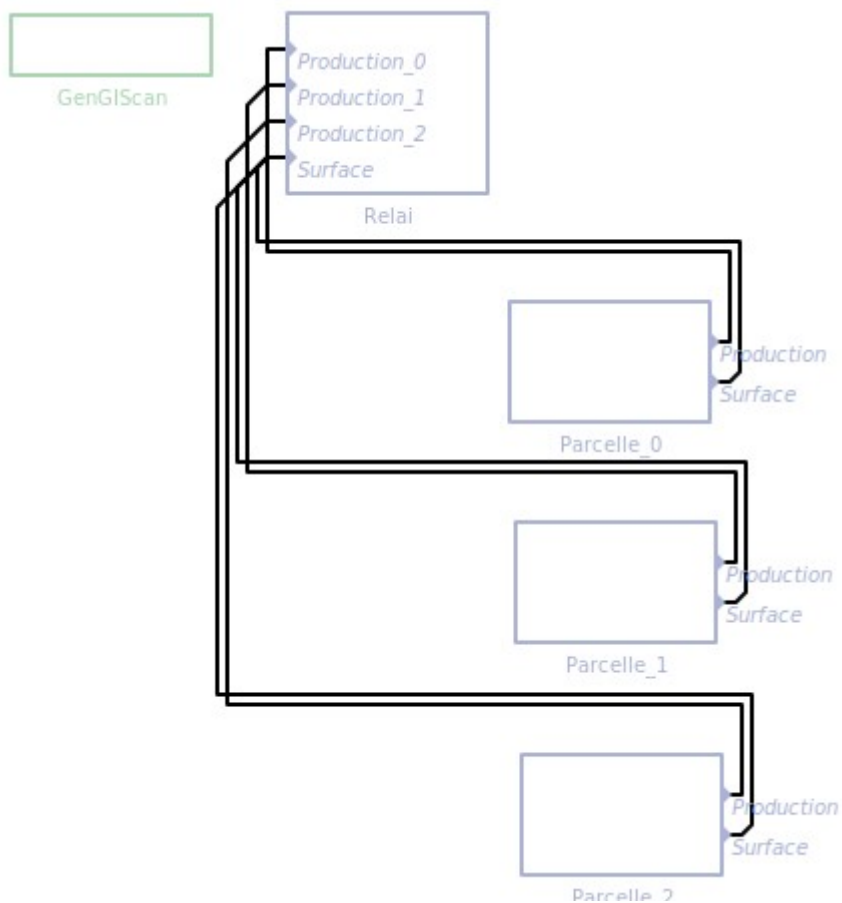
GenCSVcan PR Cl.vpz :

R_inputPortsInd > 0

R_inputPortsCom > 0

R_outputPortsInd = 0

R_outputPortsCom = 0



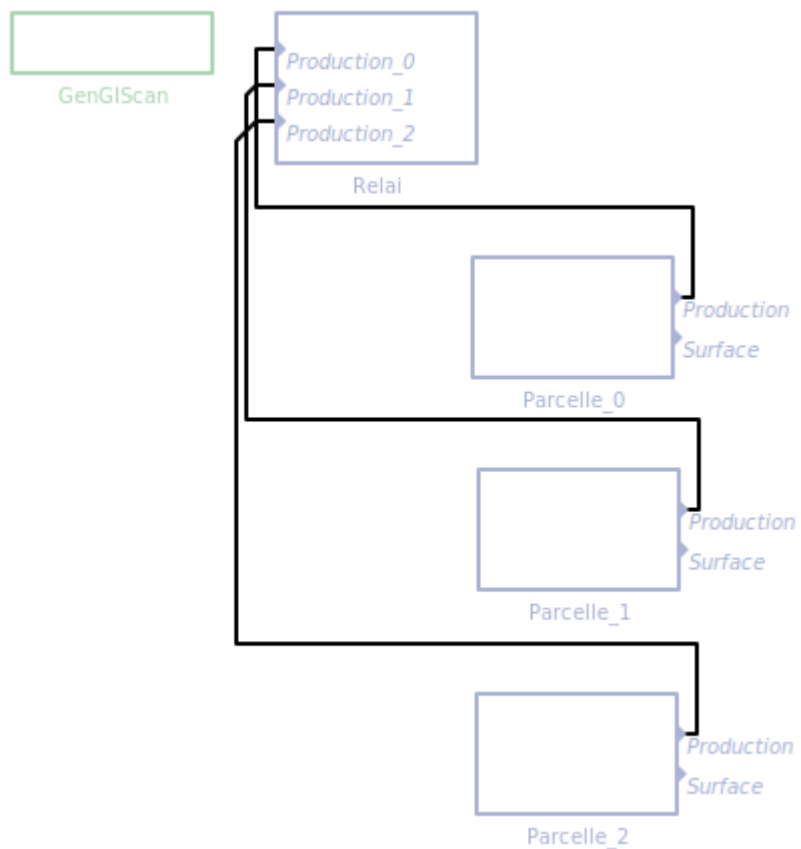
GenCSVcan PR I.vpz :

R_inputPortsInd > 0

R_inputPortsCom = 0

R_outputPortsInd = 0

R_outputPortsCom = 0



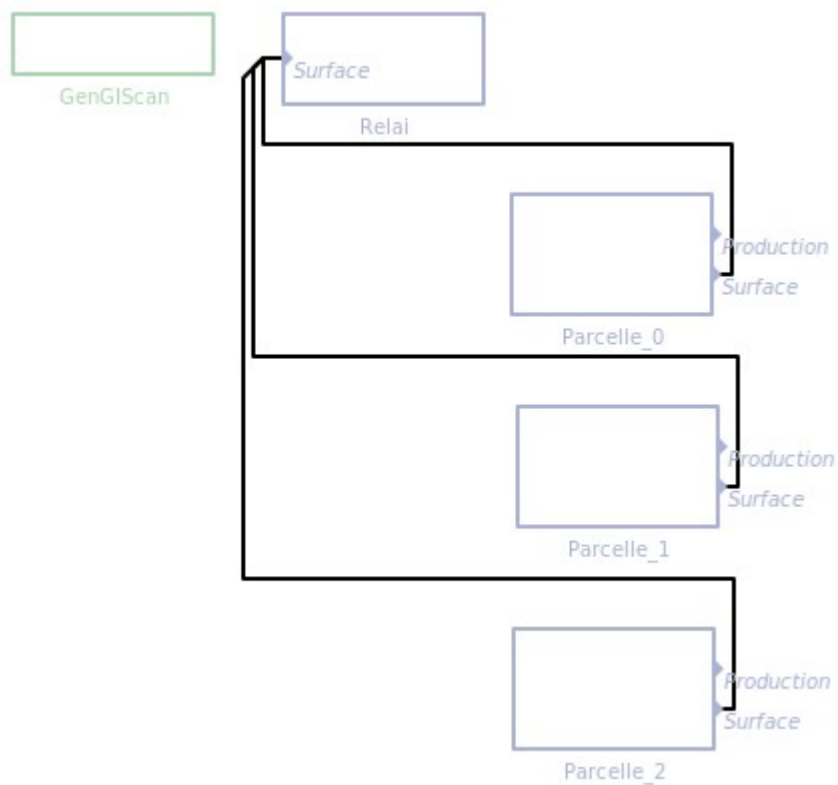
GenCSVcan PR_C.vpz :

R_inputPortsInd = 0

R_inputPortsCom > 0

R_outputPortsInd = 0

R_outputPortsCom = 0



GenCSVcan RP_XX :

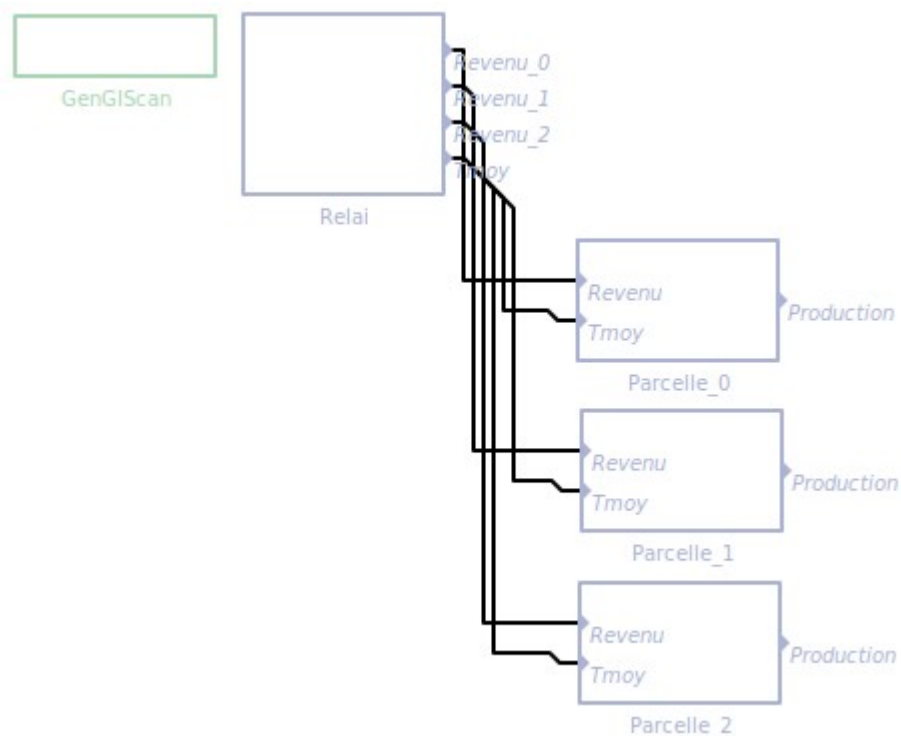
GenCSVcan RP_CI.vpz :

R_inputPortsInd = 0

R_inputPortsCom = 0

R_outputPortsInd > 0

R_outputPortsCom > 0



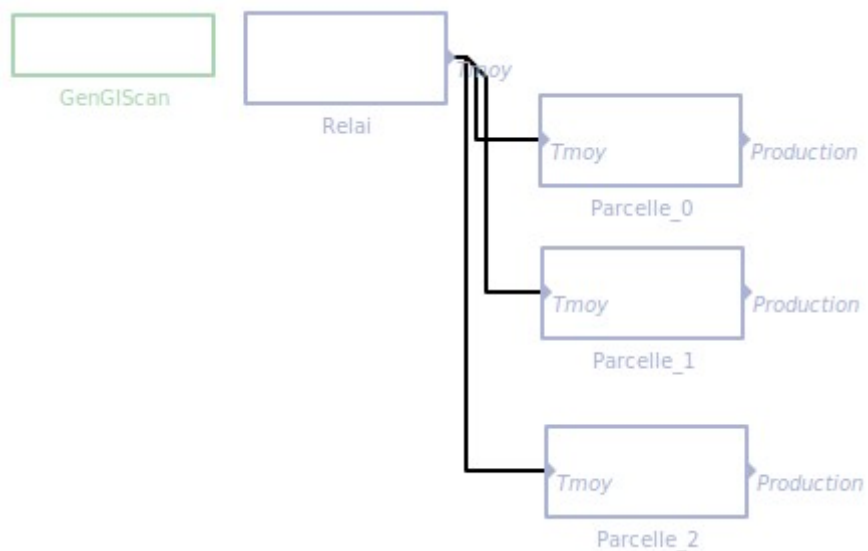
GenCSVcan RP_C.vpz :

R_inputPortsInd = 0

R_inputPortsCom = 0

R_outputPortsInd = 0

R_outputPortsCom > 0



GenCSVcan RP_I.vpz :

R_inputPortsInd = 0

R_inputPortsCom = 0

R_outputPortsInd > 0

R_outputPortsCom = 0

