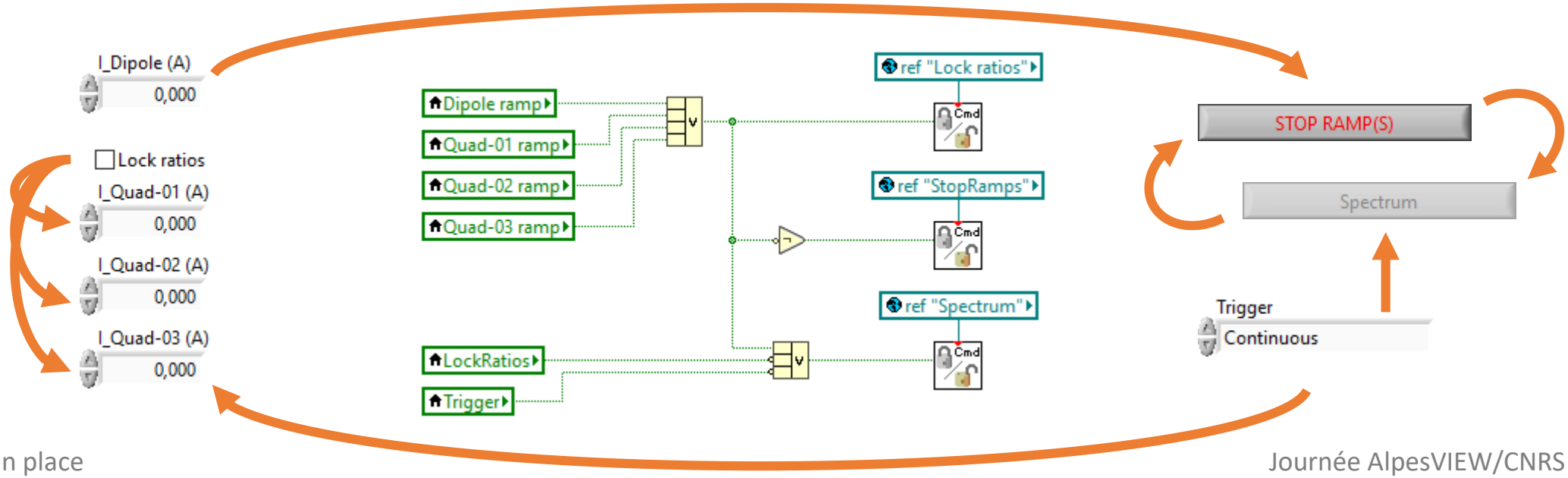
GUIInterlocks : un dispositif performant pour contrôler les verrouillages mutuels dans LabVIEW



Vocabulaire

Interlocks ⇔ Verrouillages mutuels
(de sécurité)

Vocabulaire

Dans le domaine ferroviaire :

Enclenchement

(...) ensemble d'appareils de signalisation qui matérialise physiquement, dans la zone d'action d'un poste d'aiguillage (...) une incompatibilité de manœuvre entre différents organes de commande d'appareils de voie ou de signaux dans le but de n'autoriser le passage d'un mouvement sur les différents appareils de voie que lorsque toutes les conditions de sécurité nécessaires à ce mouvement sont réalisées.

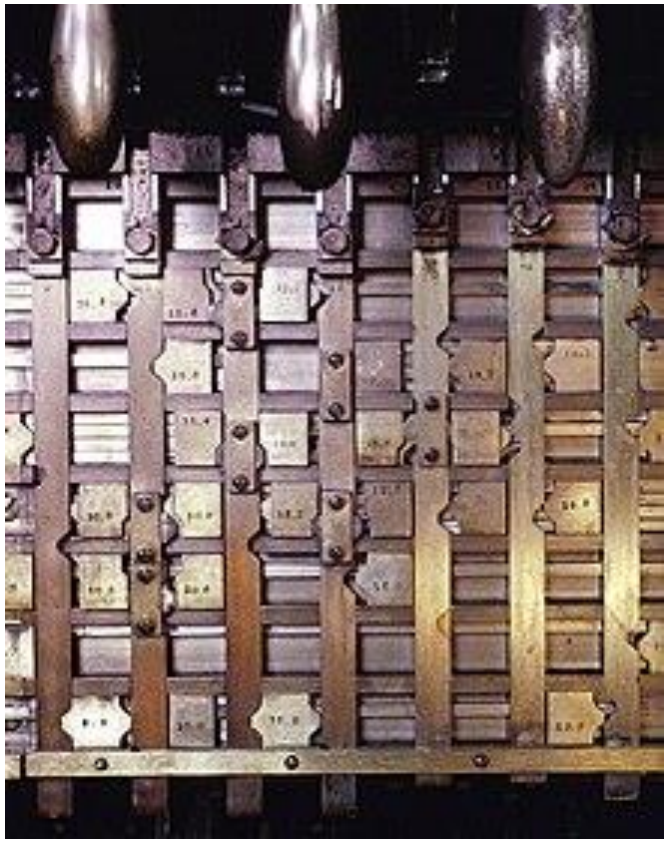


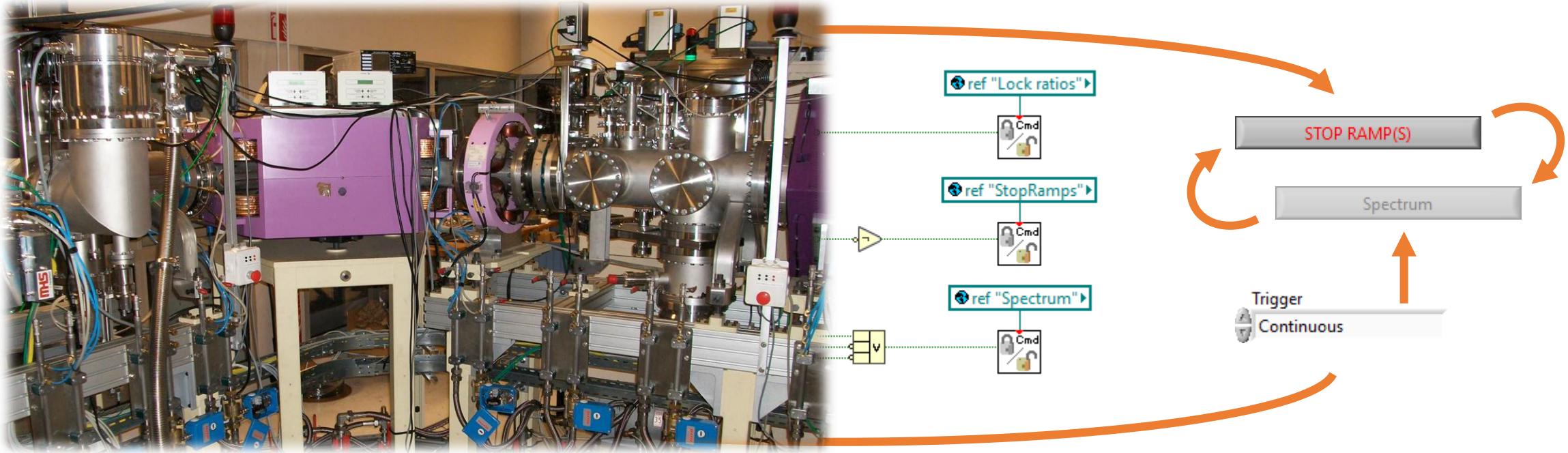
Table d'enclenchements

https://fr.wikipedia.org/wiki/Enclenchement

https://en.wikipedia.org/wiki/Interlocking

Problématique

Dans le domaine logiciel… et spécialement le contrôle-commande !

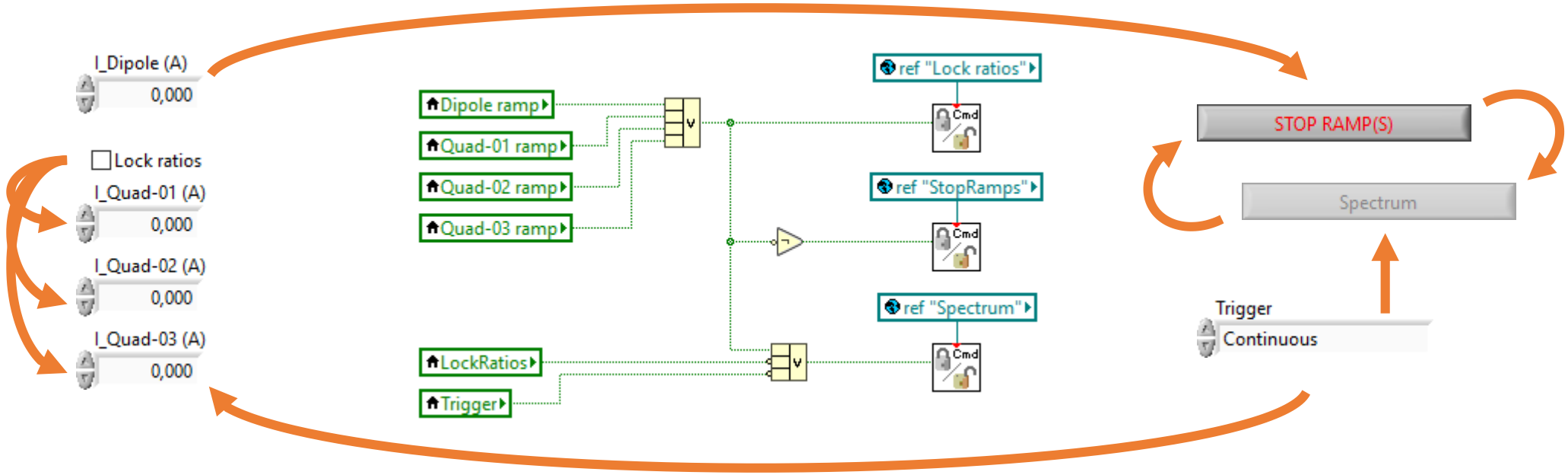


Sécurisation logicielle de l'installation + Sécurisation de l'interface utilisateur

règles de sécurité, contrôle de l'accessibilité des commandes, onglets, contrôles masqués ou désactivé+grisés…

Problématique

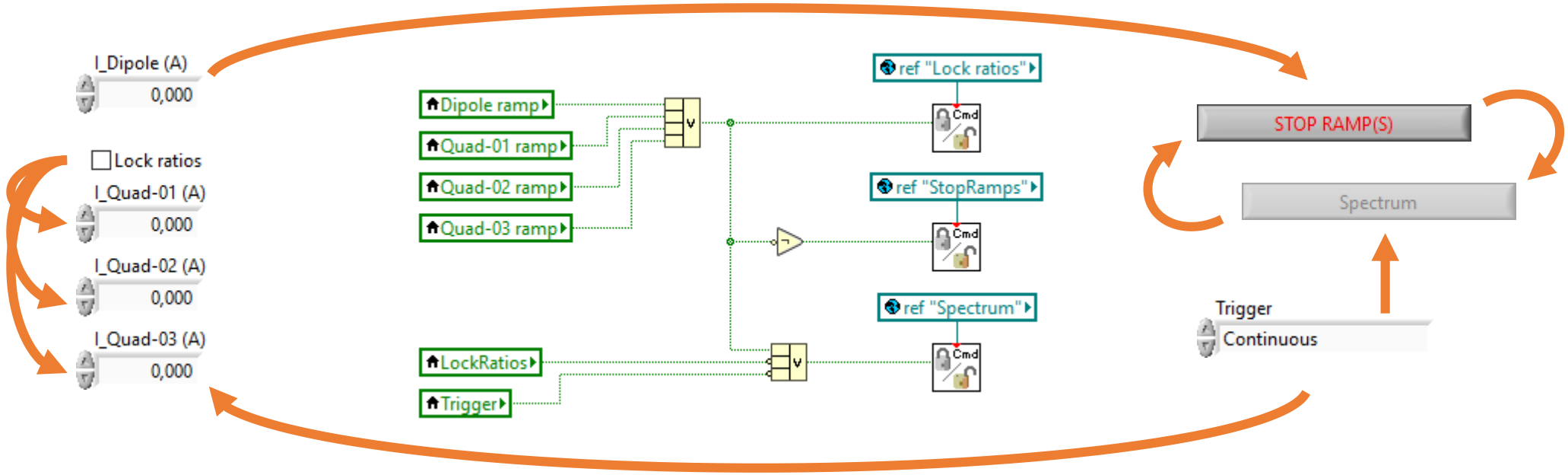
Quel modèle pour implémenter les règles de sécurité ? en LabVIEW ? en général ?



Moyens "passifs" ? Masquage, grisage ? Où placer le code qui gère les verrouillages ? Distribué au plus proche de ses objets ou centralisé ? Quand évaluer les règles logiques ? Comment limiter le couplage avec les autres facettes du logiciel ?...

*** GUIInterlocks ***

Un design pattern efficace ; des VIs modèles à adapter dans votre appli LabVIEW.

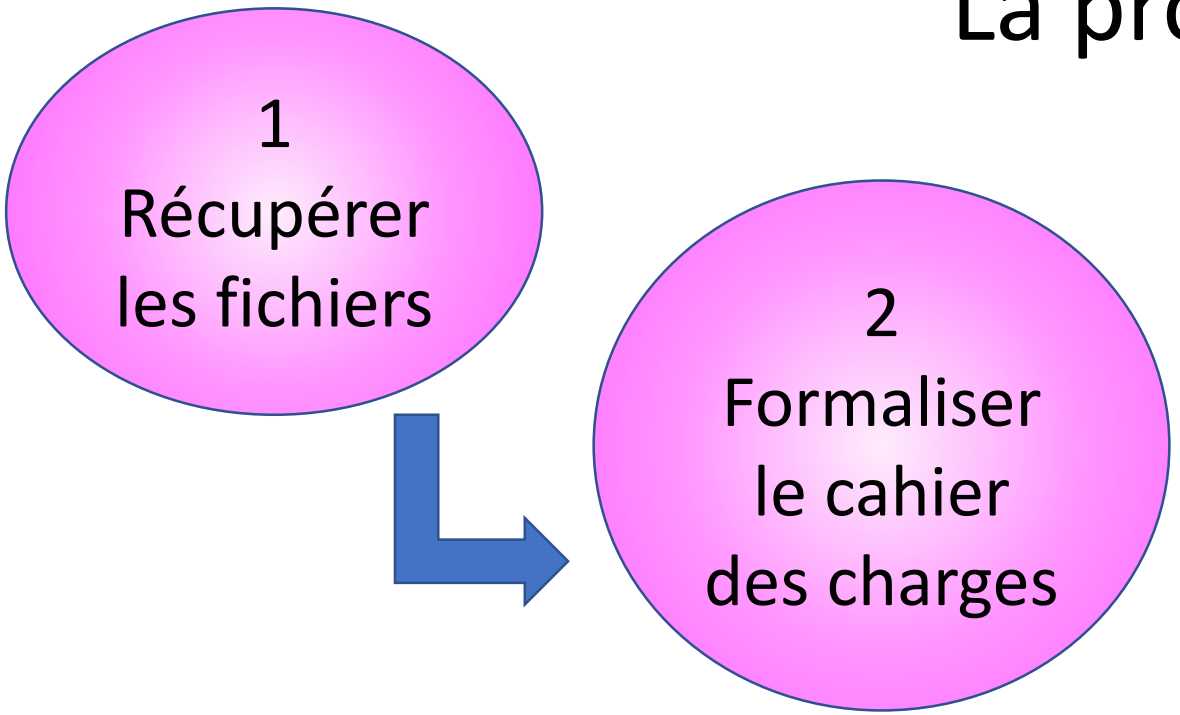


https://gitlab.in2p3.fr/oz-reuse/labview/ozr-GUIInterlocks

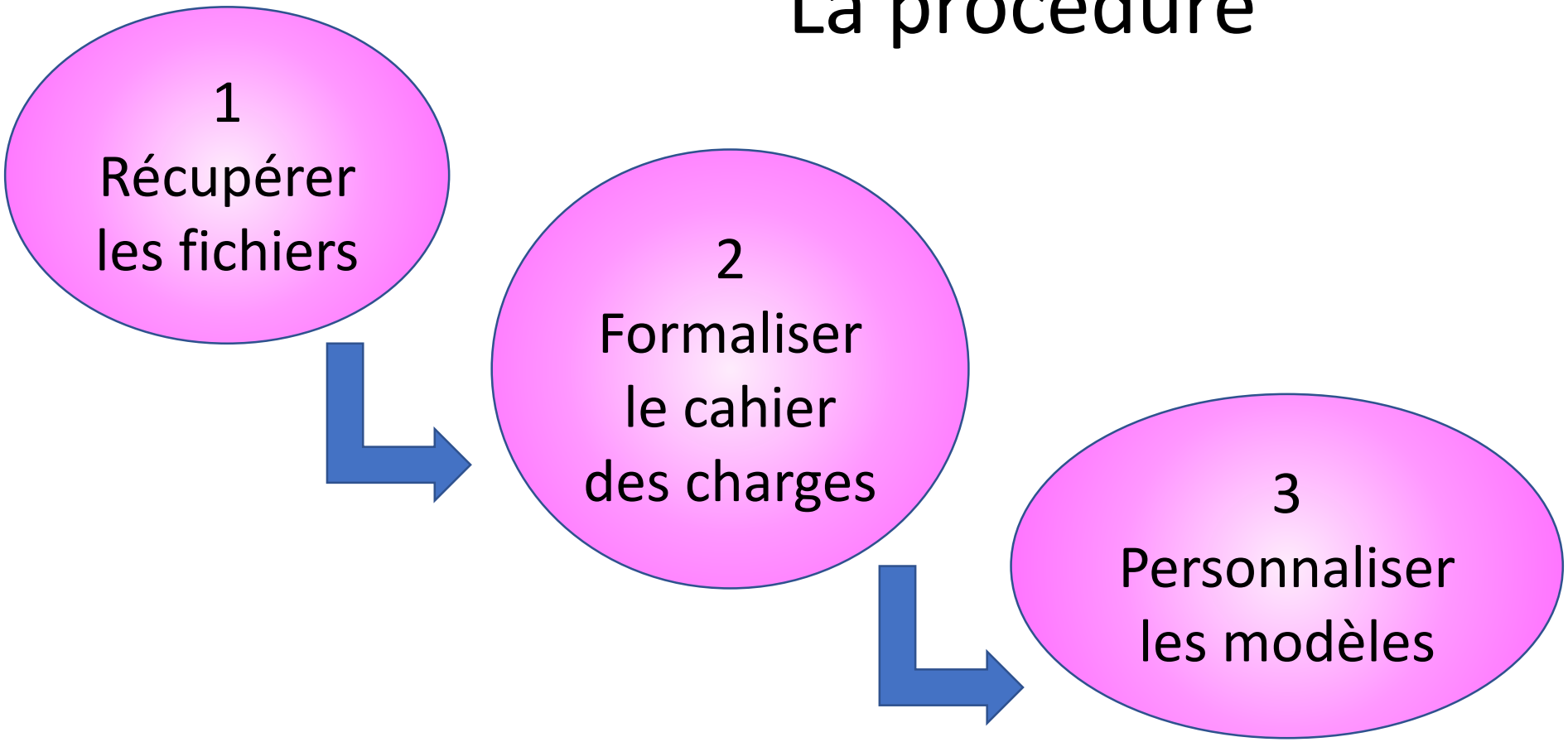
La procédure

1
Récupérer
les fichiers

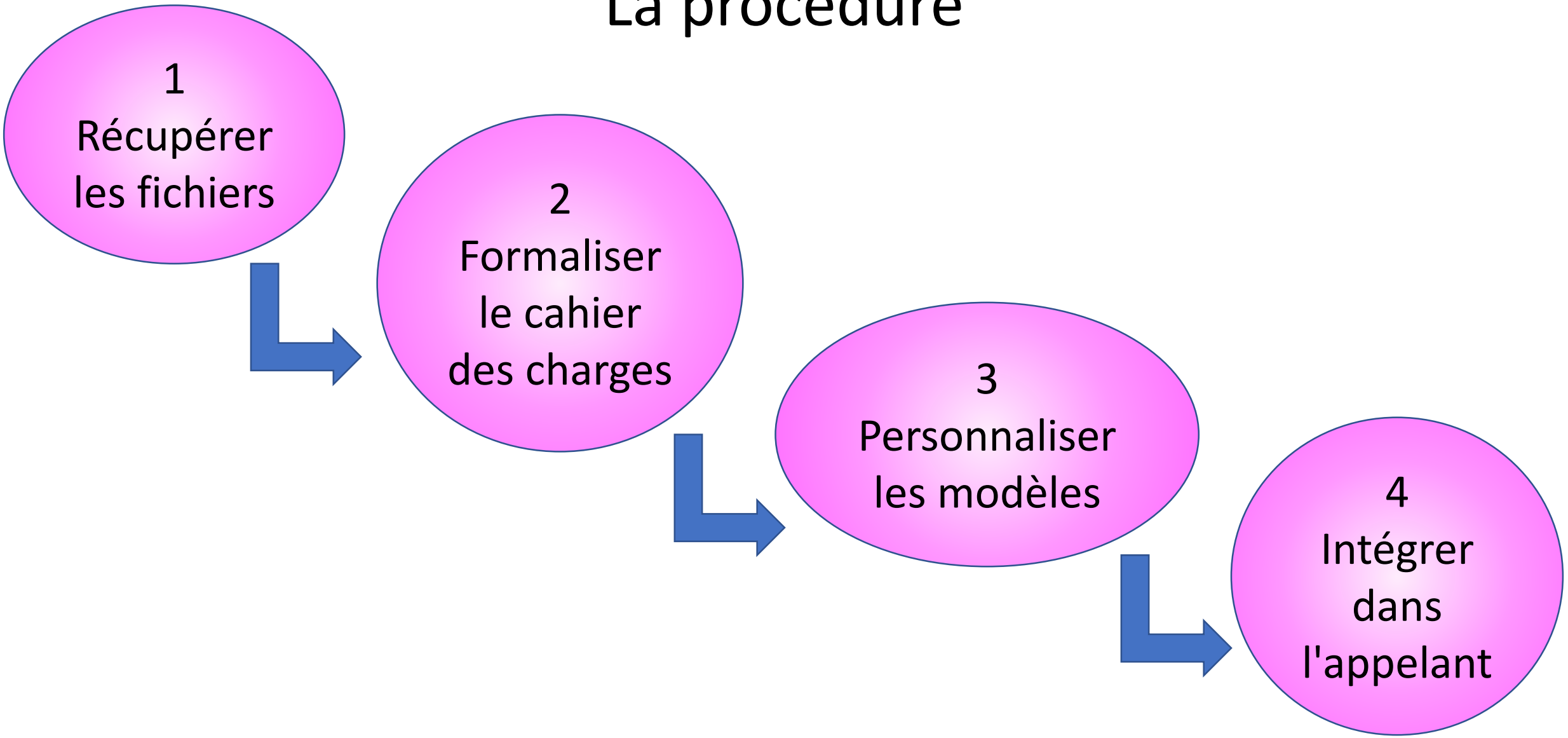
La procédure



La procédure



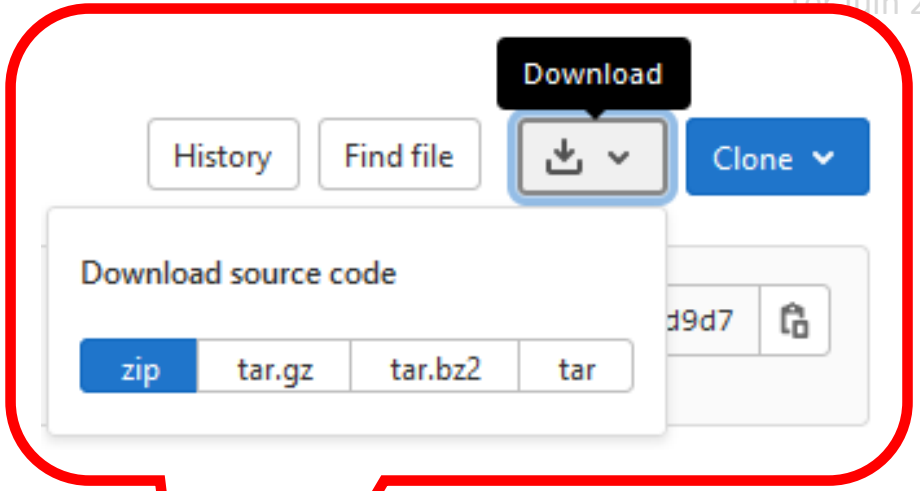
La procédure

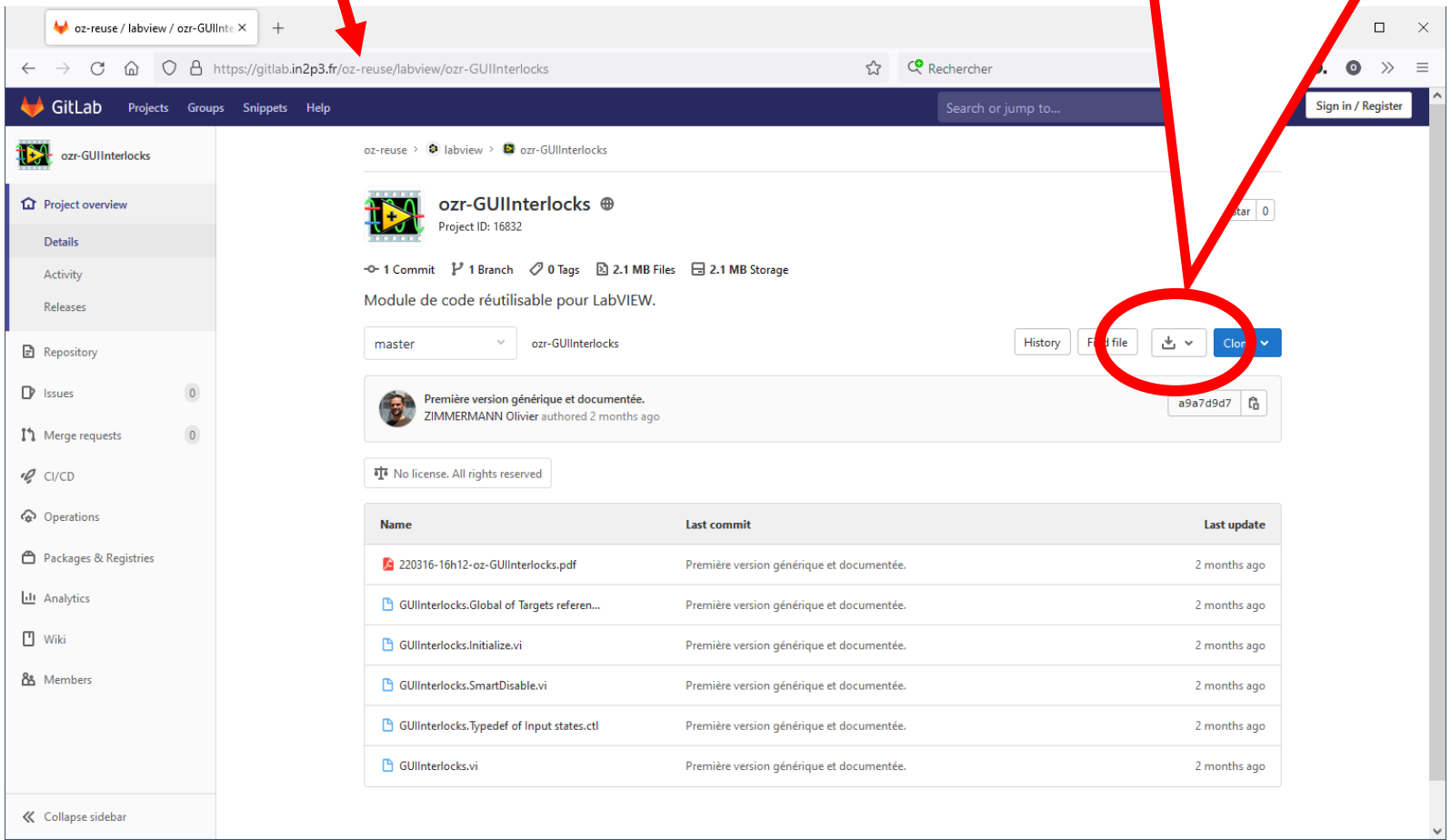


1.
Récupérer les fichiers

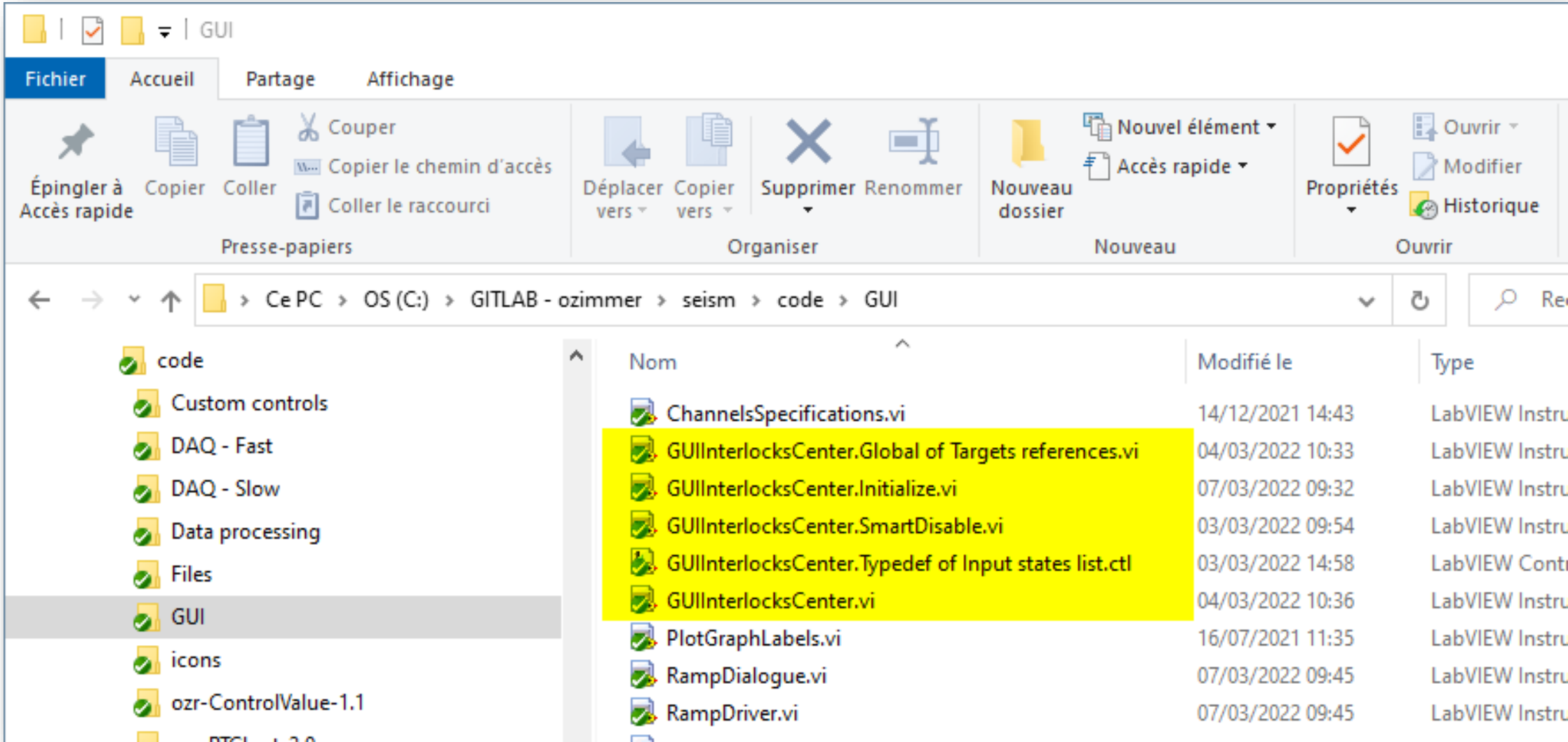
1.1 Récupérer en ligne les fichiers modèles

https://gitlab.in2p3.fr/oz-reuse/labview/ozr-GUIInterlocks



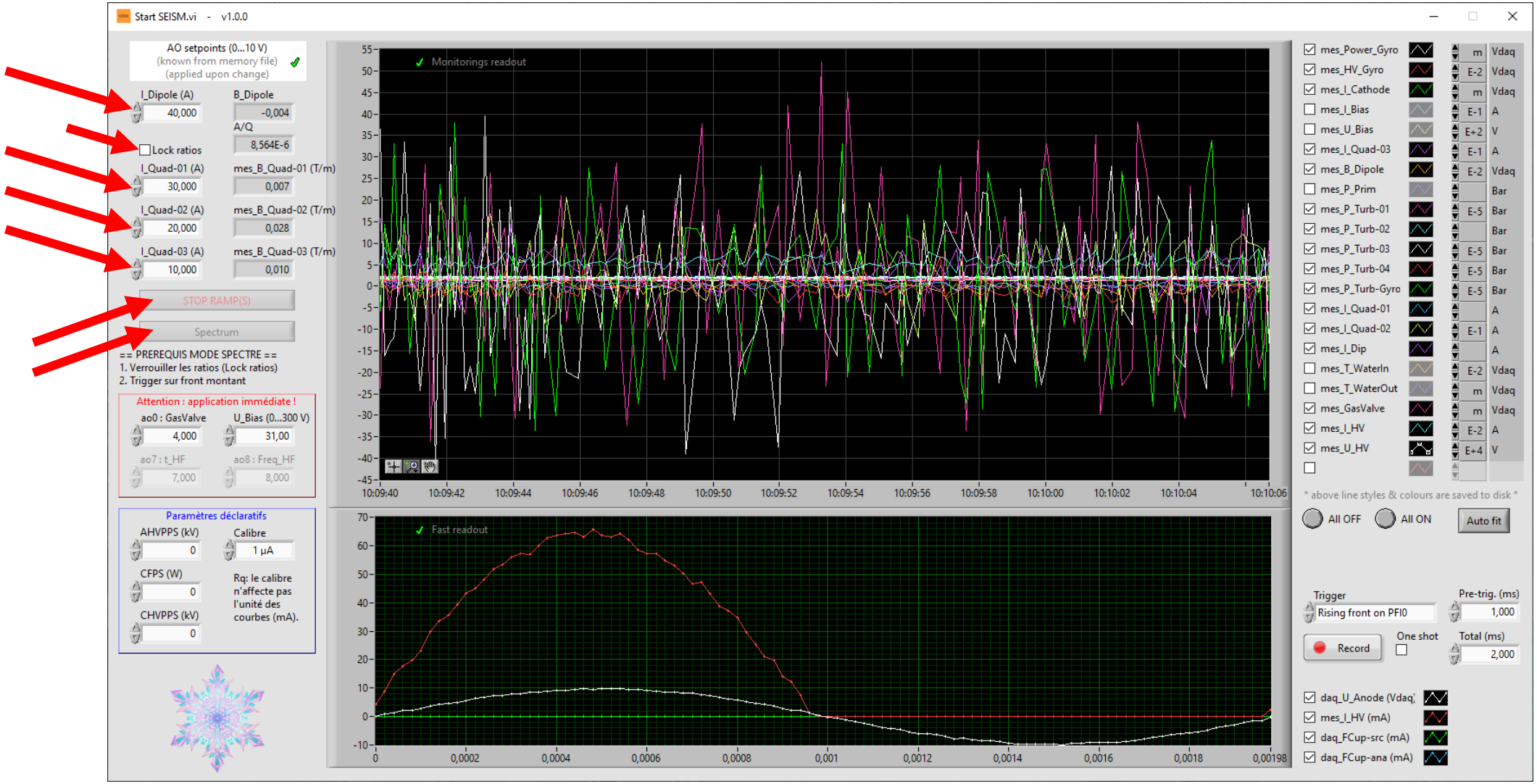


1.2 Ajoutez les 5 fichiers modèles du dispositif dans votre code source.

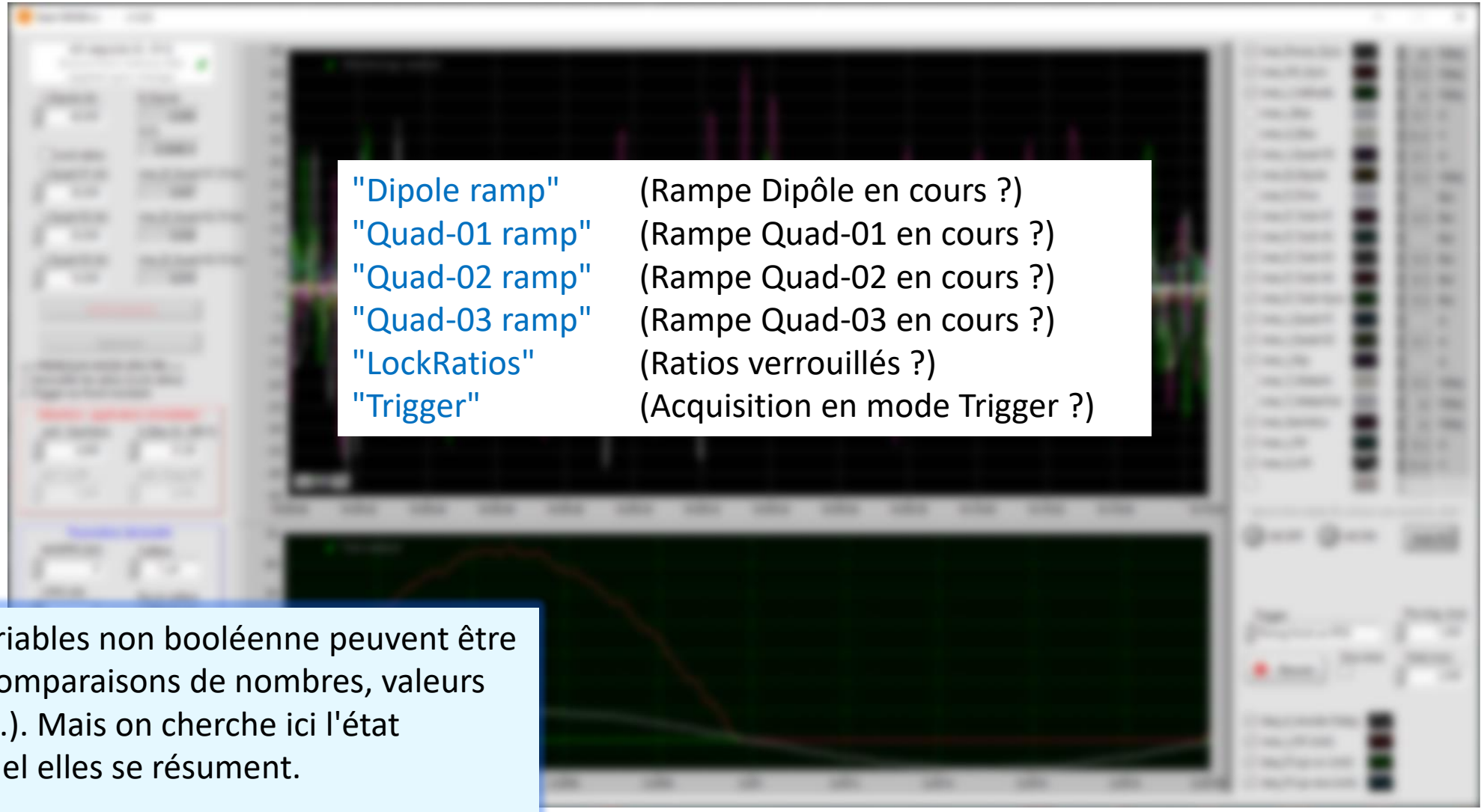


2.
Identification du cahier des charges dans le programme

2.1 Listez les commandes ou indicateurs qui doivent obéir à des règles de (dé)verrouillage (***états de sortie***).

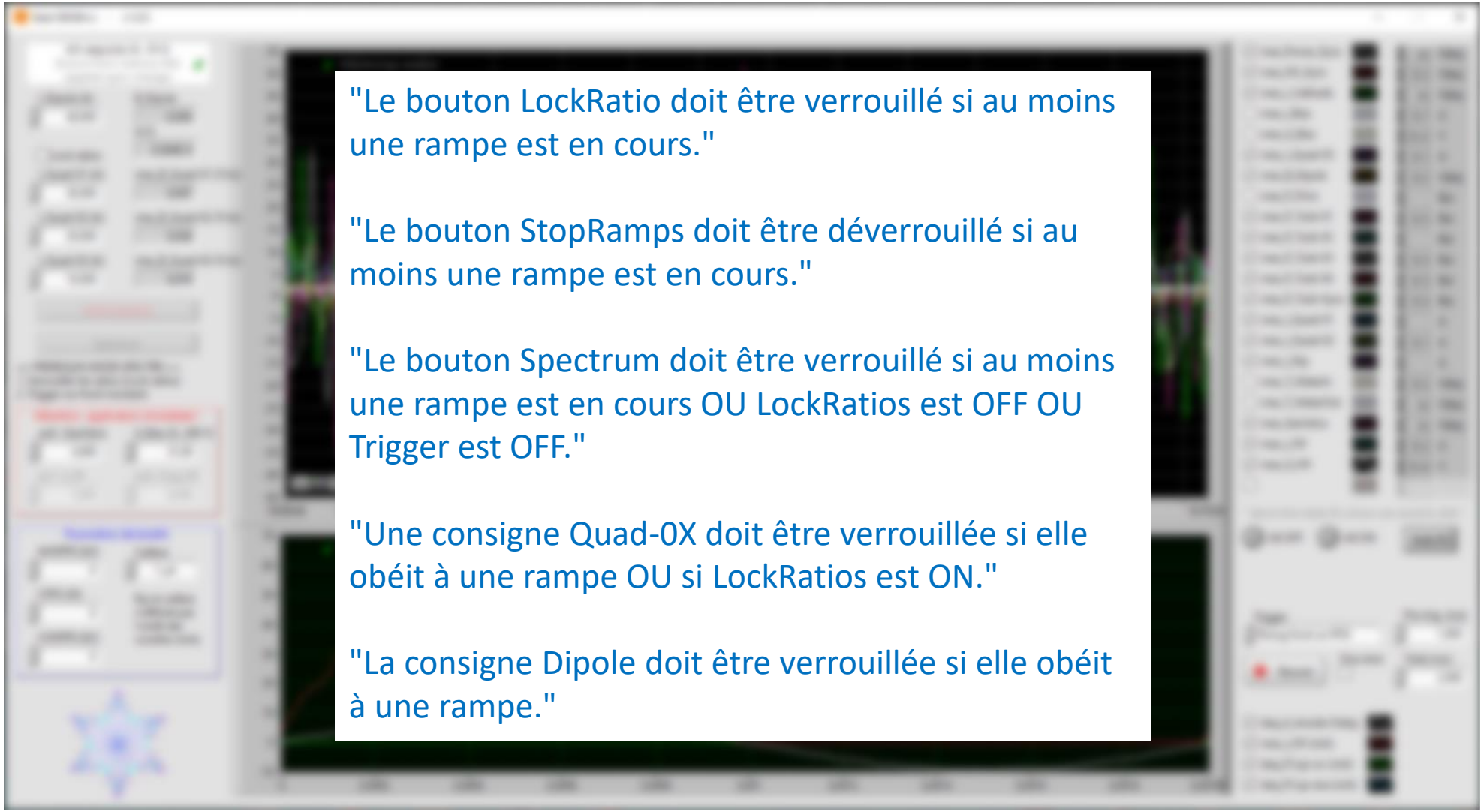


2.2 Identifiez et nommez les états booléens dont dépendent les règles de (dé)verrouillage (***états d'entrée***).



Note : Des variables non booléenne peuvent être impliquées (comparaisons de nombres, valeurs particulières...). Mais on cherche ici l'état booléen auquel elles se résument.

2.3 Ecrivez sous forme logique les différentes ***règles de verrouillage*** reliant les états d'entrée aux états de sortie.



"Le bouton LockRatio doit être verrouillé si au moins une rampe est en cours."

"Le bouton StopRamps doit être déverrouillé si au moins une rampe est en cours."

"Le bouton Spectrum doit être verrouillé si au moins une rampe est en cours OU LockRatios est OFF OU Trigger est OFF."

"Une consigne Quad-0X doit être verrouillée si elle obéit à une rampe OU si LockRatios est ON."

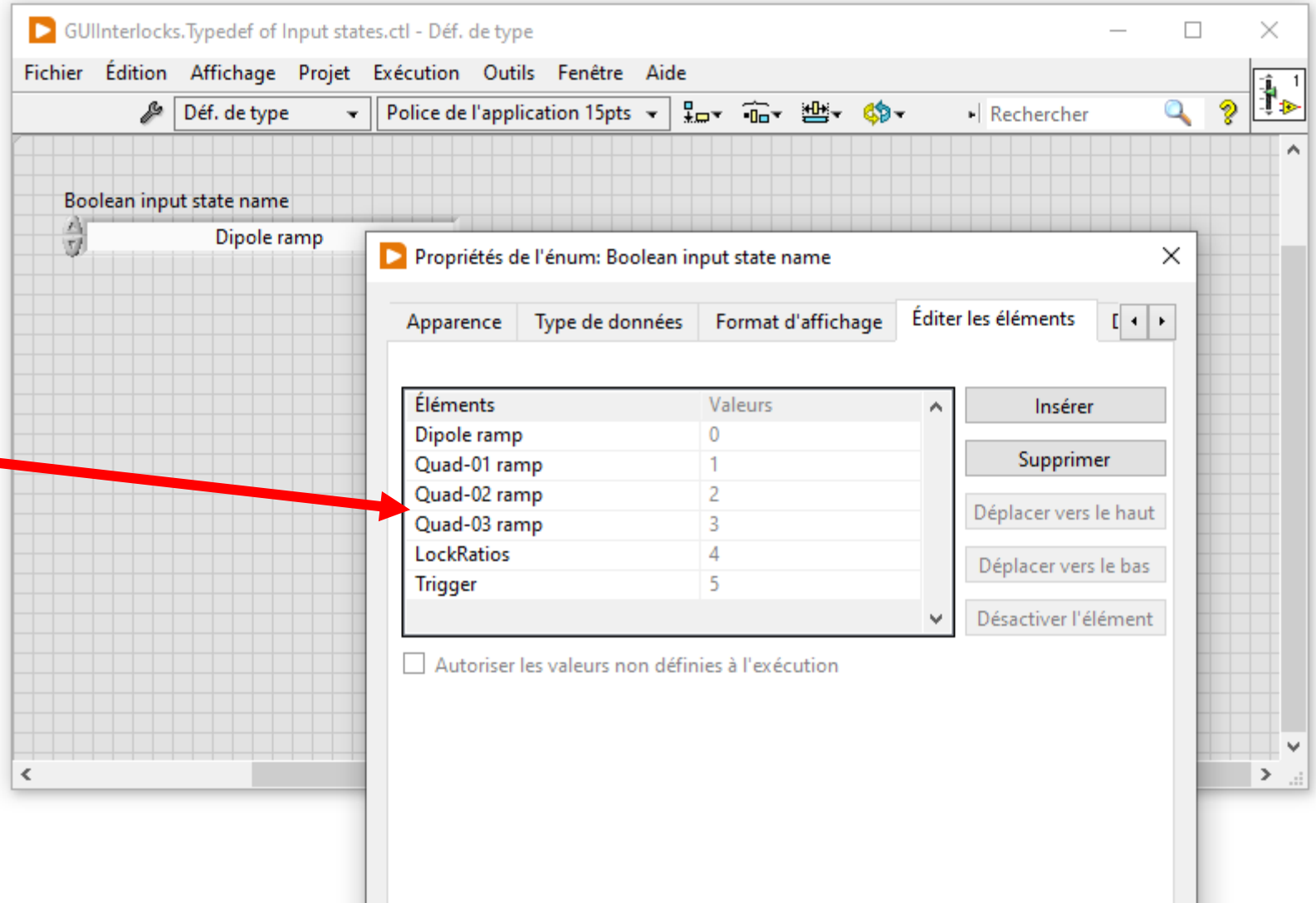
"La consigne Dipole doit être verrouillée si elle obéit à une rampe."

3.
Personnalisation des fichiers modèles

3.1 Typedef of Input states list.ctl

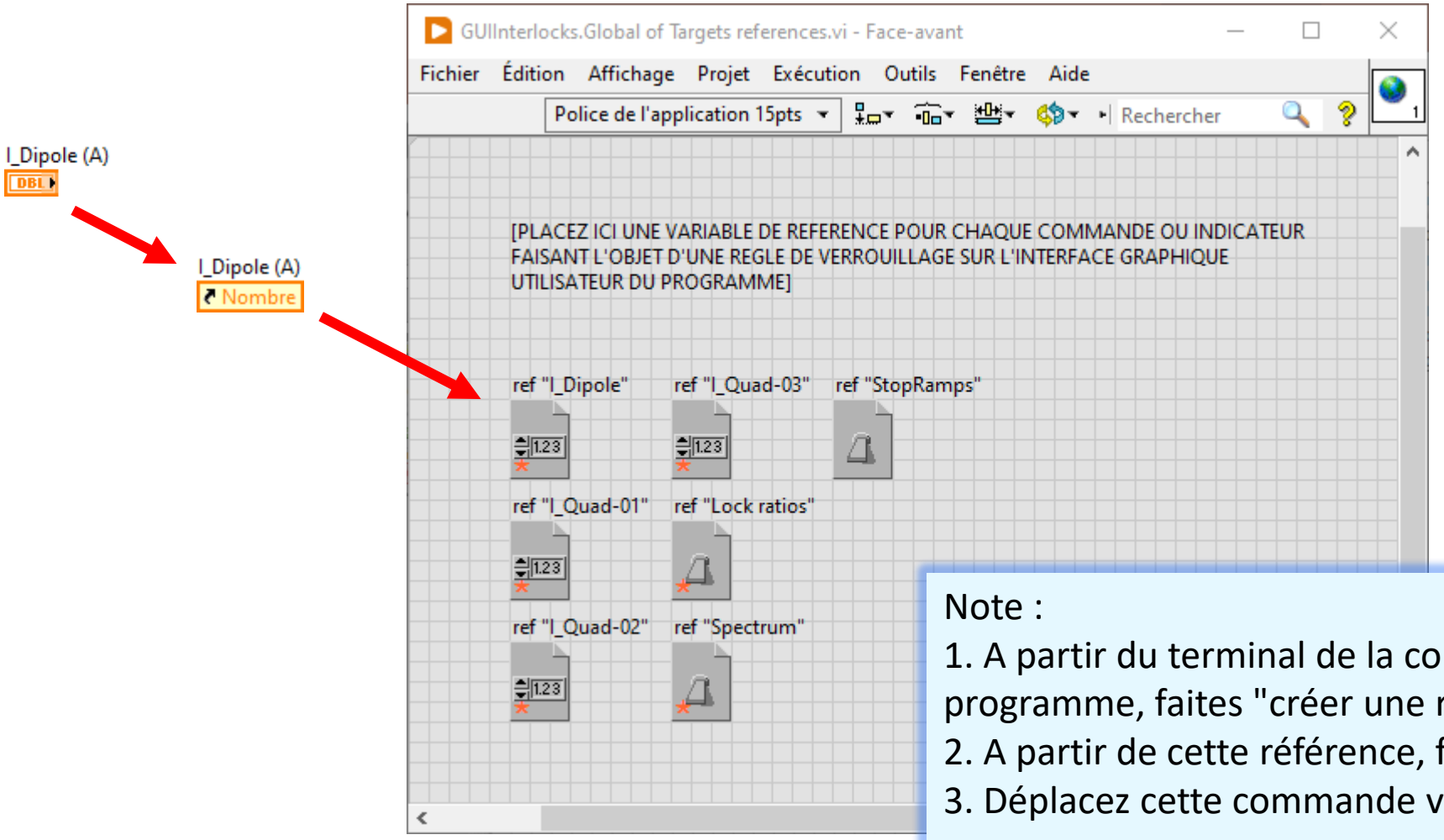
Saisissez la liste des états d'entrée.

"Dipole ramp"
"Quad-01 ramp"
"Quad-02 ramp"
"Quad-03 ramp"
"LockRatios"
"Trigger"



3.2 Global of Targets references.vi

Créez un indicateur global pour la référence de chaque commande ou indicateur à (dé)verrouiller.

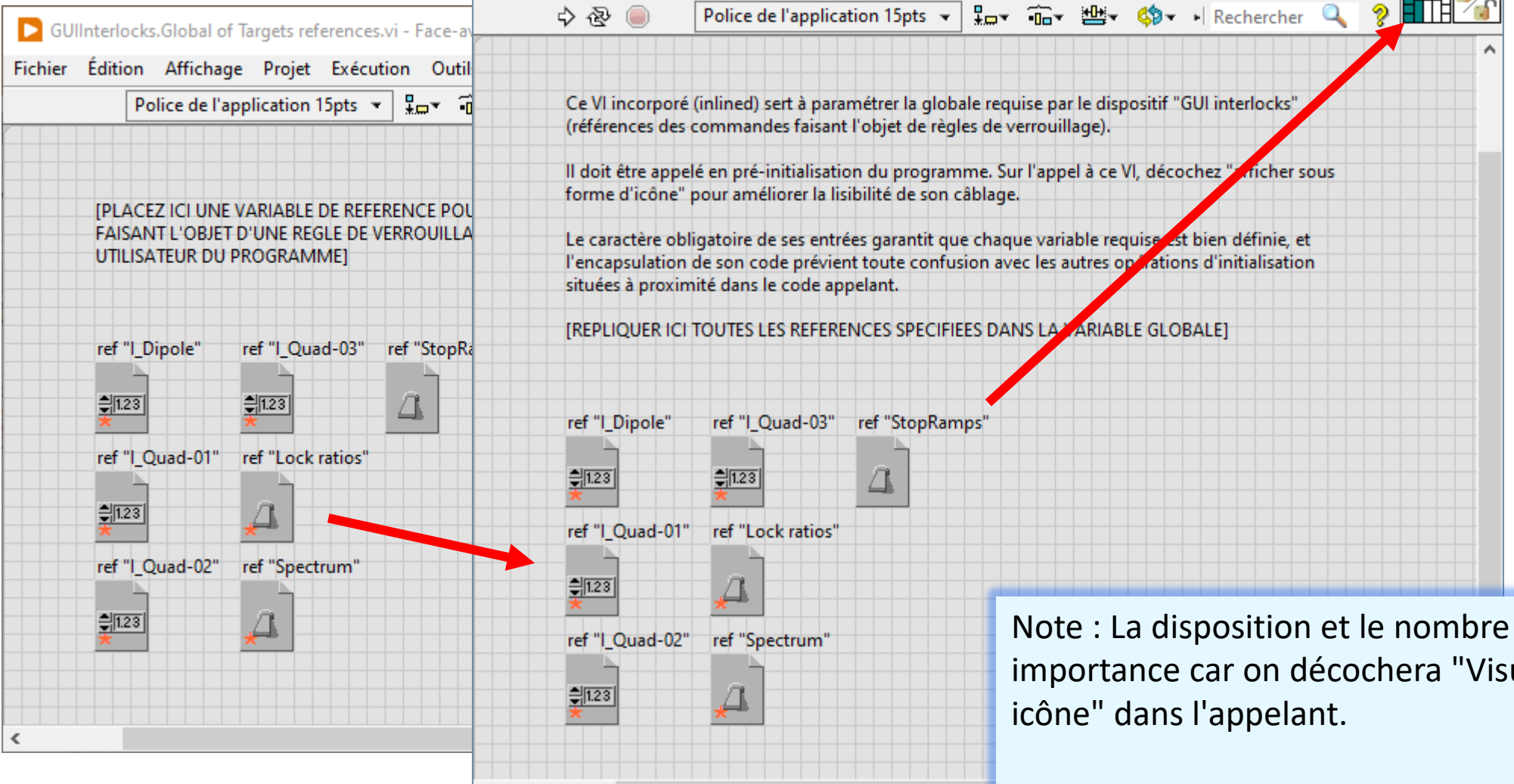


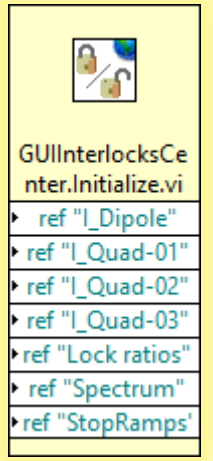
Note :

1. A partir du terminal de la commande dans le code du programme, faites "créer une référence".
2. A partir de cette référence, faites "créer une commande".
3. Déplacez cette commande vers le VI global.

3.3 **Initialize.vi** (face avant)

Répliquez l'ensemble des commandes du VI global, et associez-les aux connexions du VI.

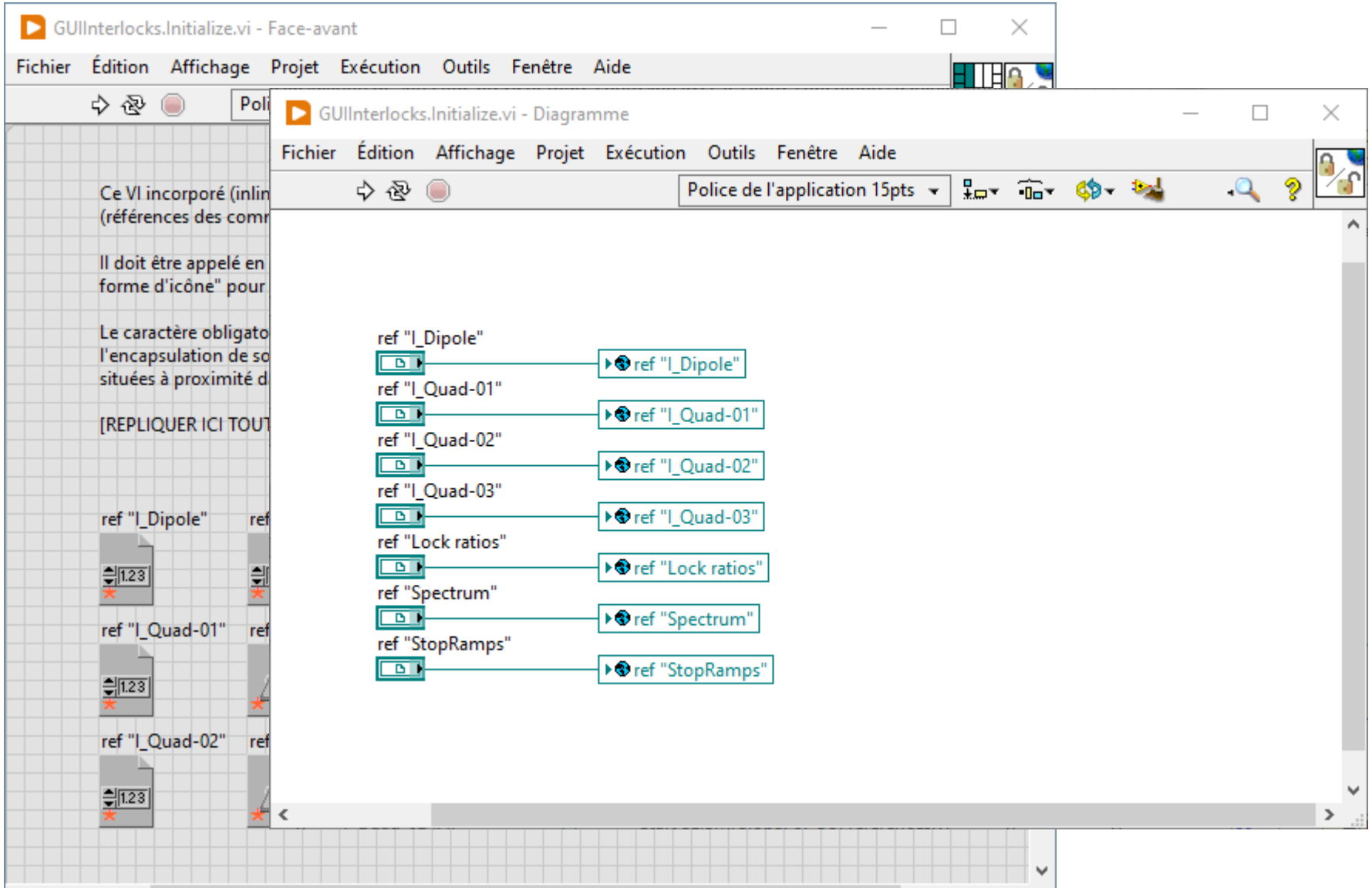




Note : La disposition et le nombre des connexions sont sans importance car on décochera "Visualiser sous la forme d'une icône" dans l'appelant.

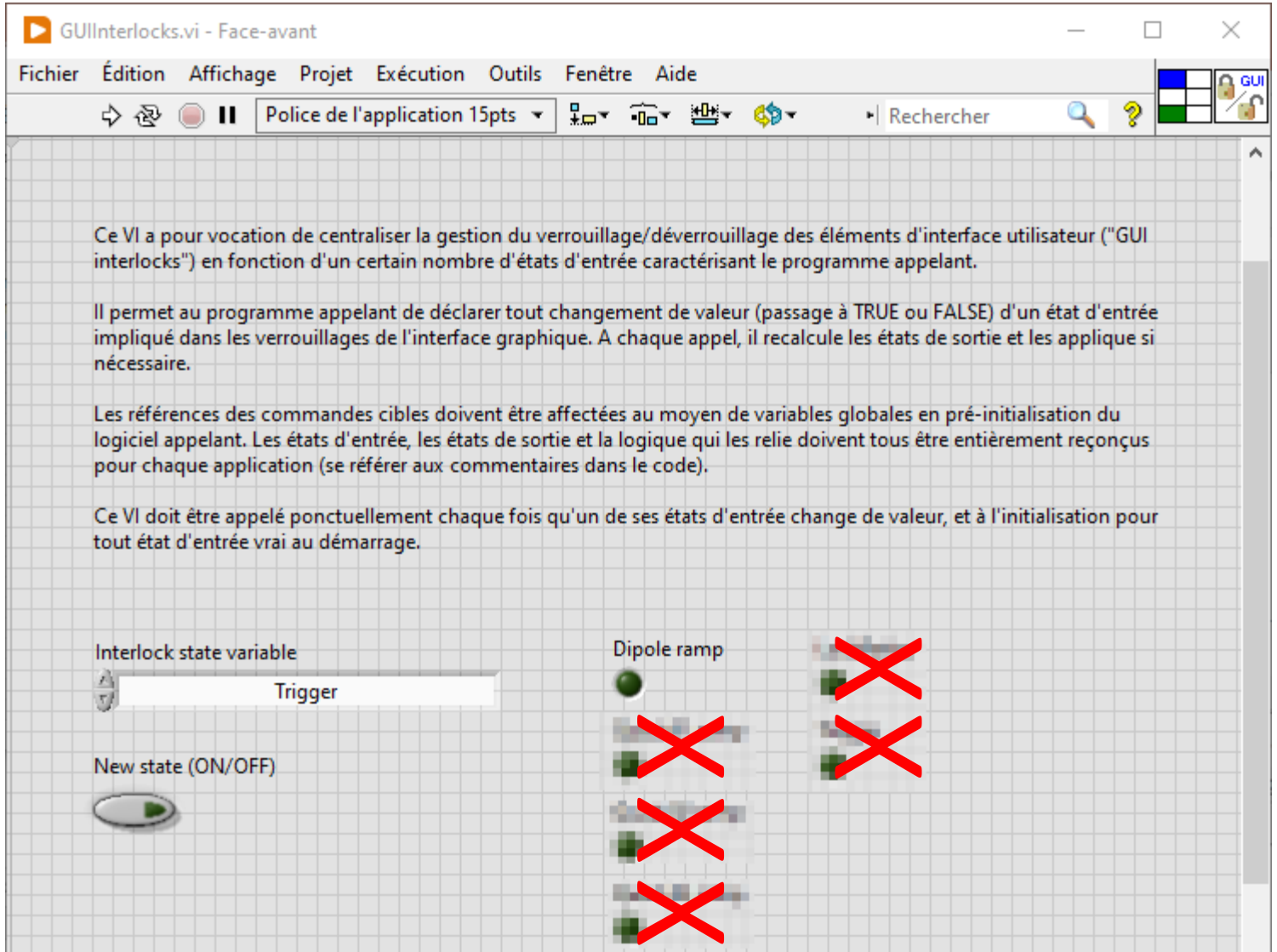
3.4 **Initialize.vi** (diagramme)

Câblez la valeur de chaque entrée vers la globale qui lui correspond.



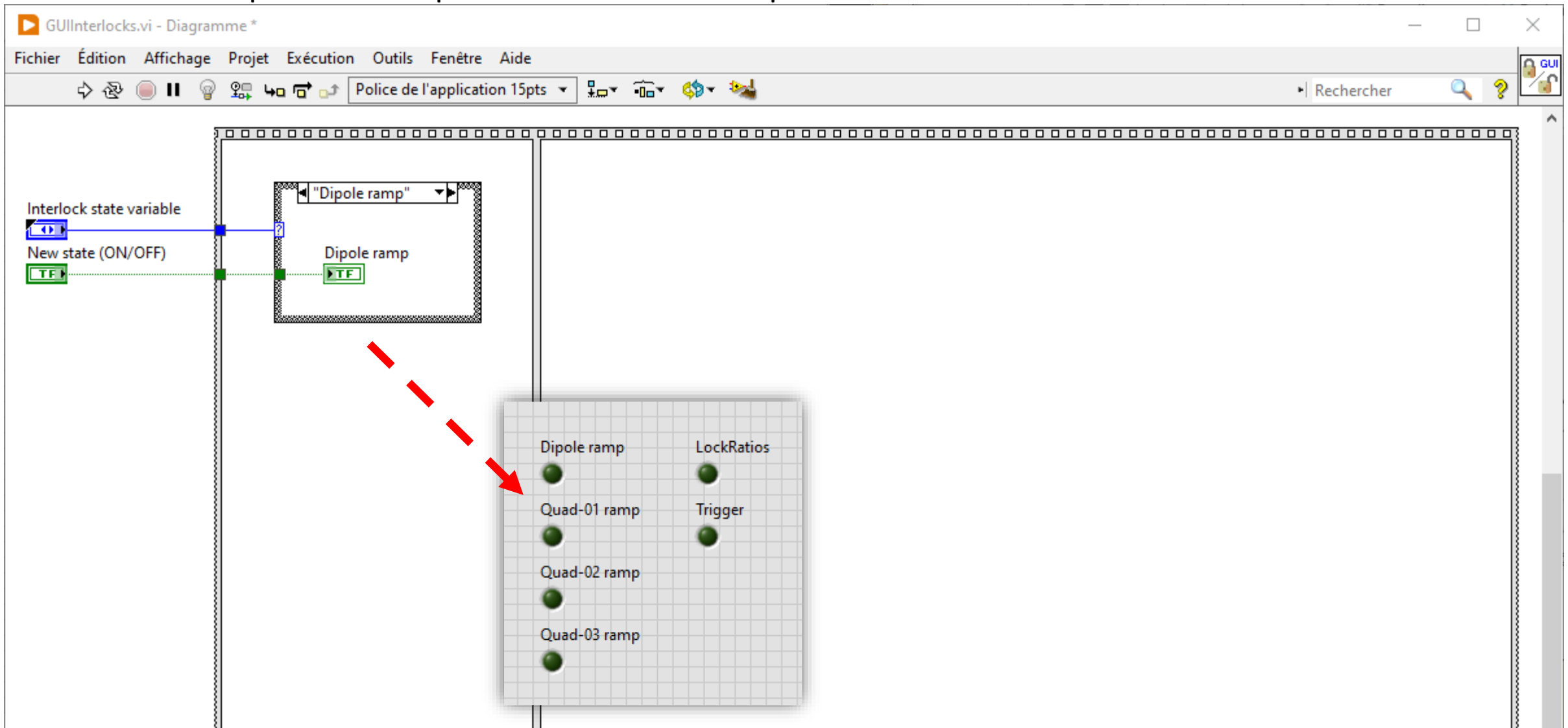
3.5 **GUIInterlocks.vi** (face avant)

Supprimez tous les indicateurs booléens du modèle sauf un et donnez-lui le nom du premier état d'entrée.



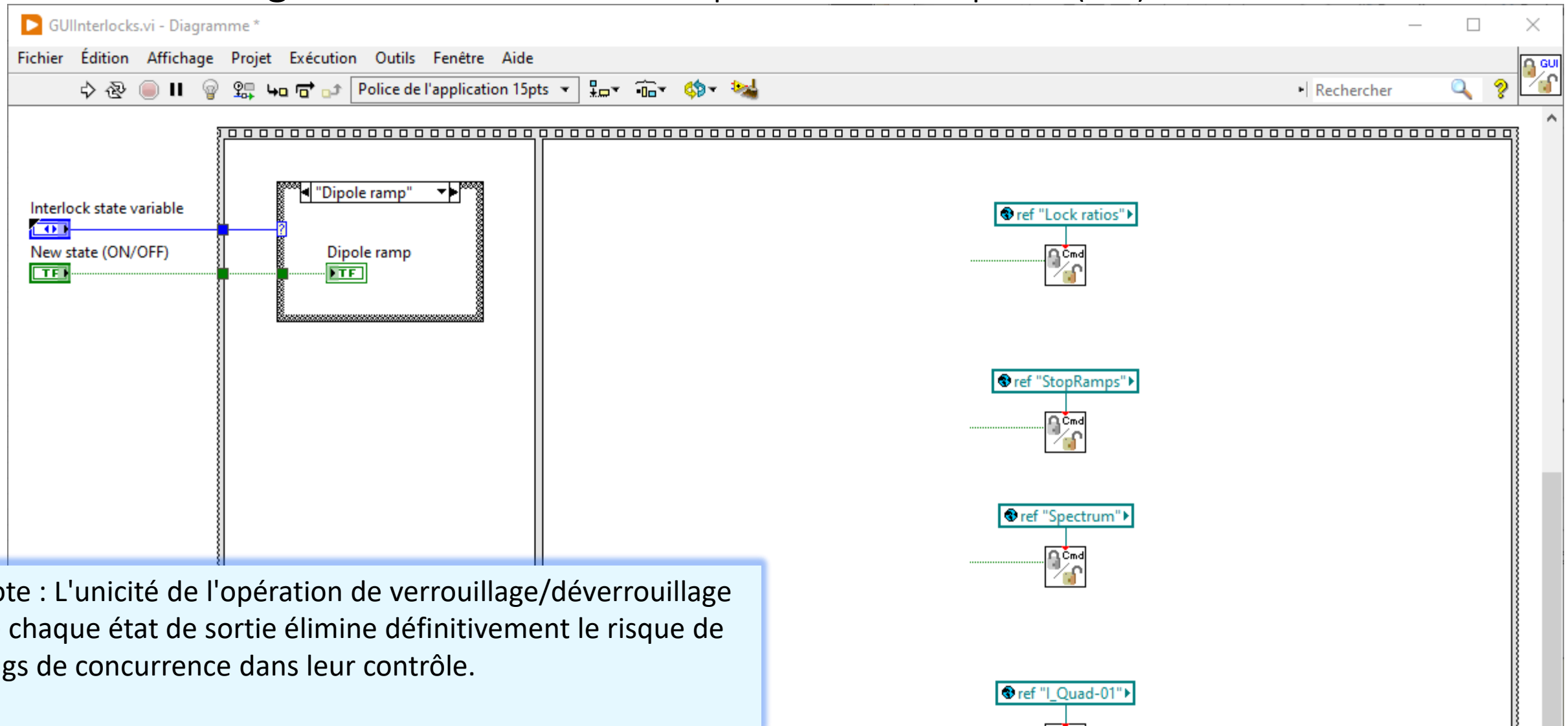
3.6 **GUIInterlocks.vi** (diagramme, à gauche)

Supprimez les cas vides de la structure conditionnelle et dupliquez le cas restant pour chaque état d'entrée possible. Renommez les indicateurs.



3.7 **GUIInterlocks.vi** (diagramme, à droite)

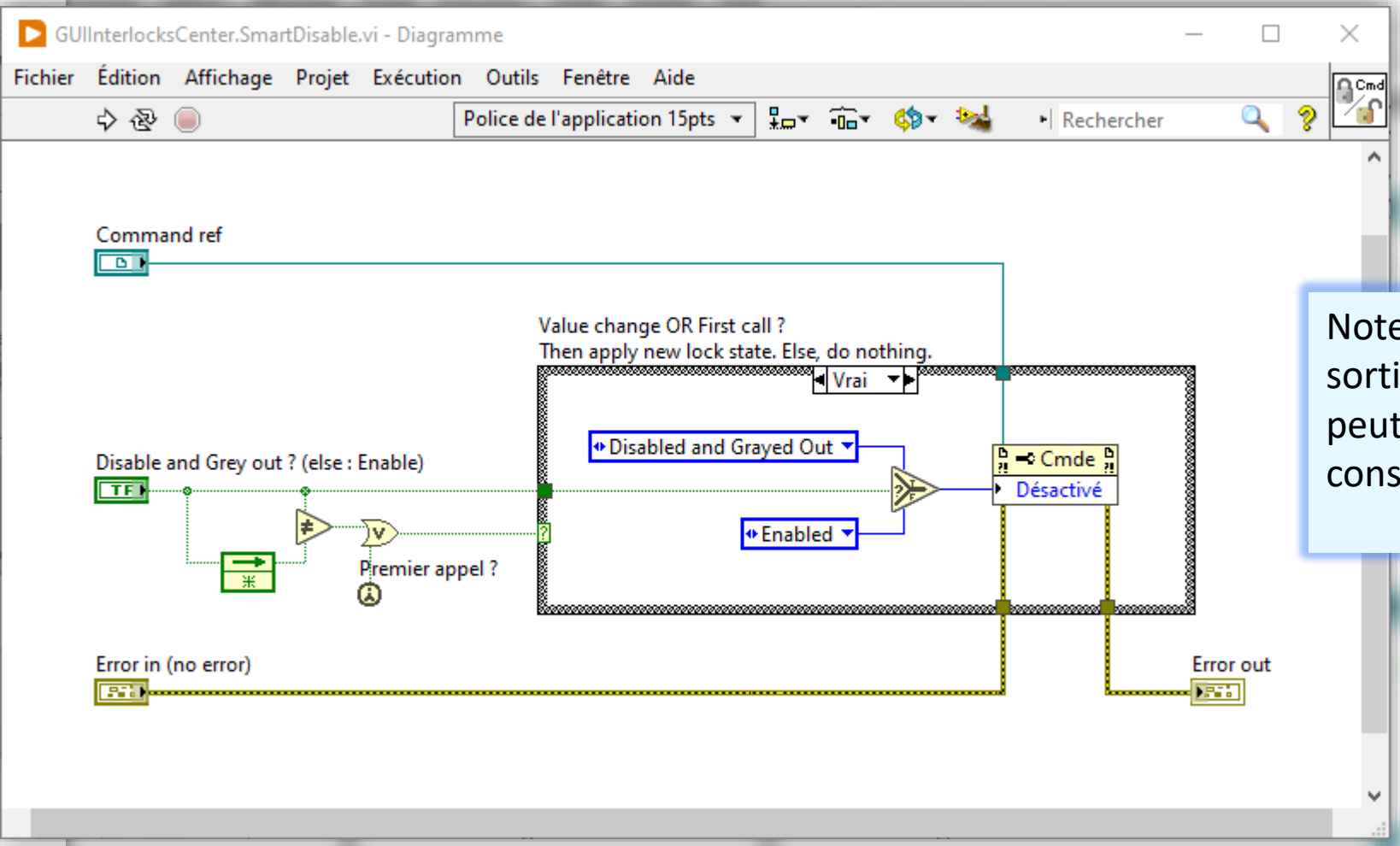
Chaque référence cible doit apparaitre ici sous la forme d'une et une seule variable globale en lecture. Adaptez le code qui la (dé)verrouille.



Note : L'unicité de l'opération de verrouillage/déverrouillage de chaque état de sortie élimine définitivement le risque de bugs de concurrence dans leur contrôle.

3.7 **GUIInterlocks.vi** (diagramme, à droite)

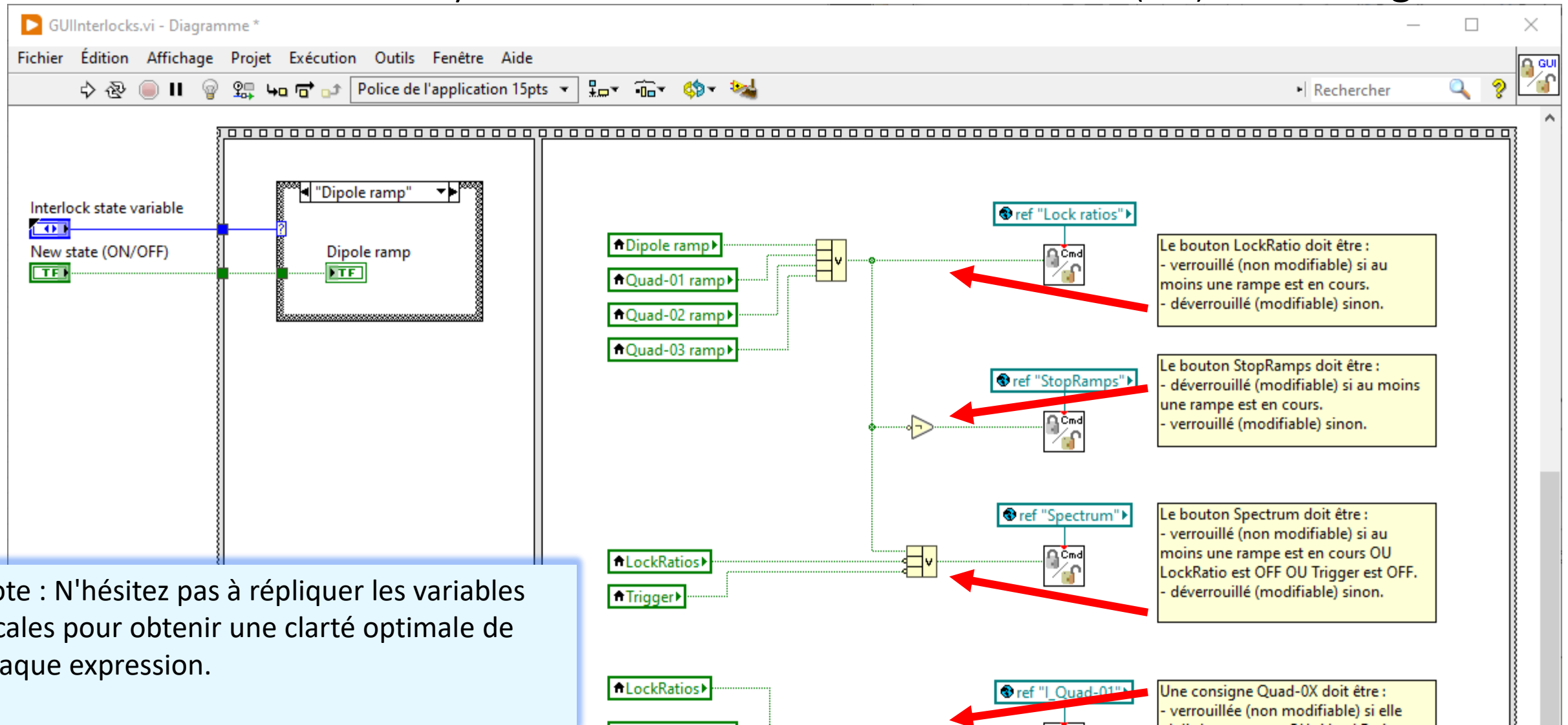
Chaque référence cible doit apparaitre ici sous la forme d'une et une seule variable globale en lecture. Adaptez le code qui la (dé)verrouille.



Note : Le risque de concurrence sur les états de sortie étant éliminé, le code de (dé)verrouillage peut être conçu pour n'opérer que si sa consigne change (ici au moyen d'un sous-VI).

3.8 **GUIInterlocks.vi** (diagramme, au centre)

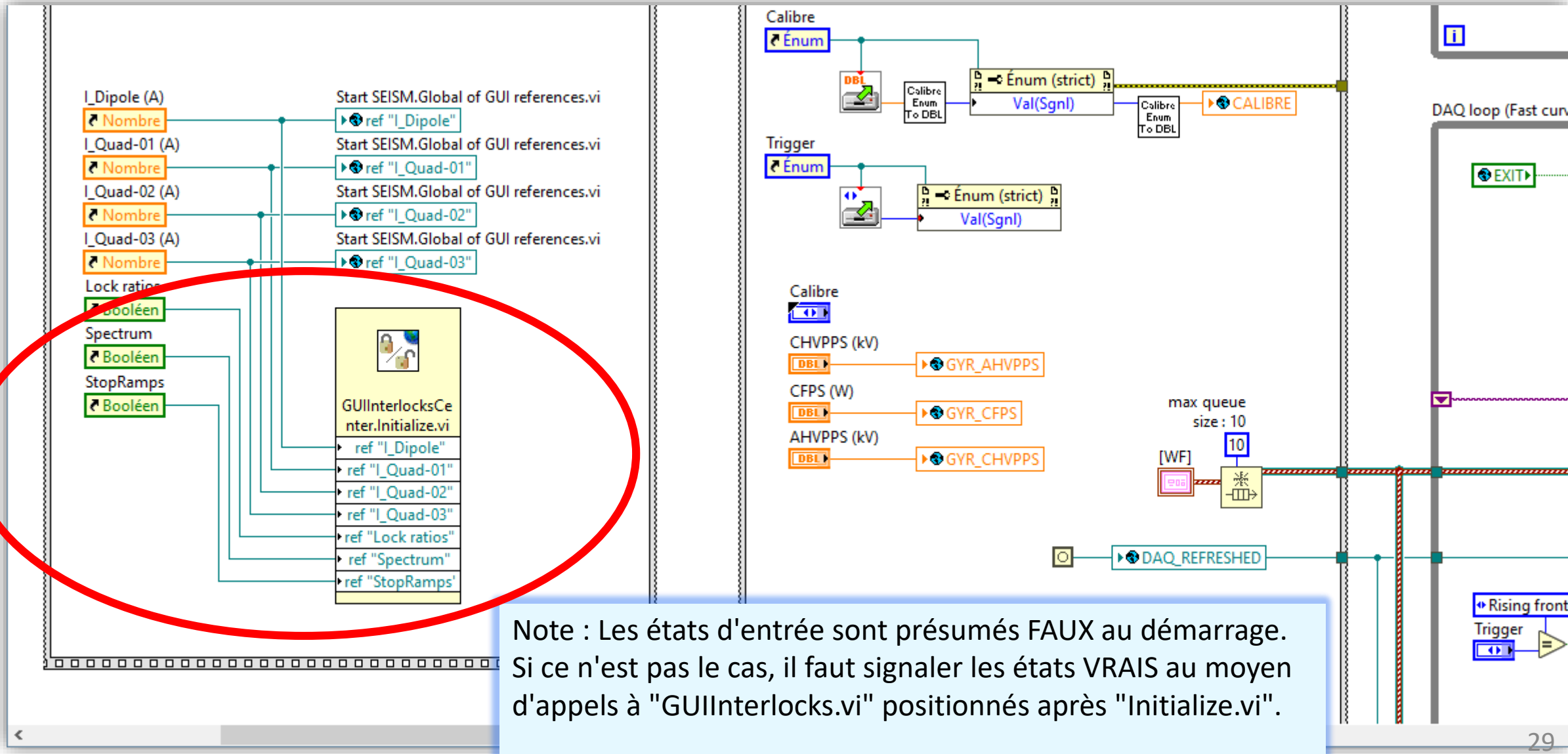
Implémentez les expressions logiques en les alimentant avec des variables locales et en envoyant les résultats sur les actions de (dé)verrouillage.



Note : N'hésitez pas à répliquer les variables locales pour obtenir une clarté optimale de chaque expression.

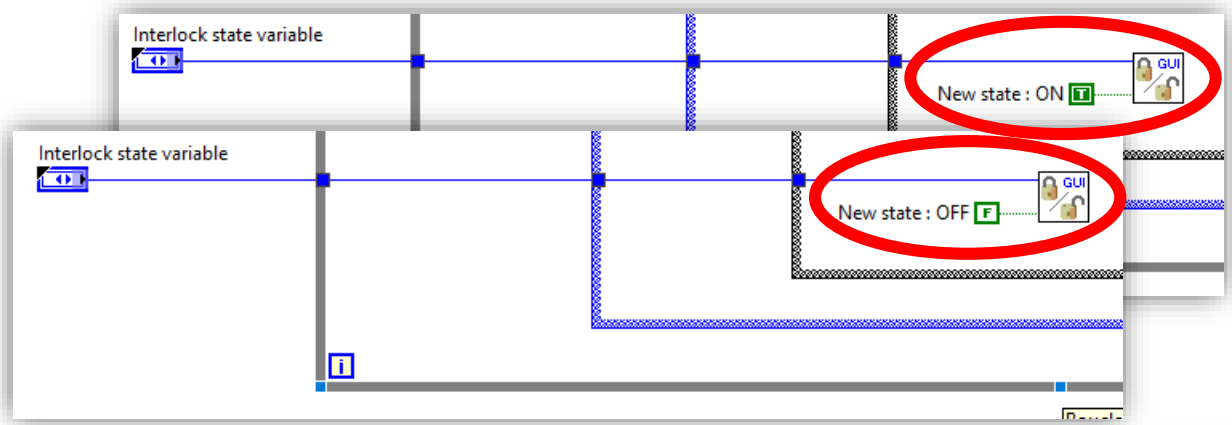
4.
Intégration au programme appelant

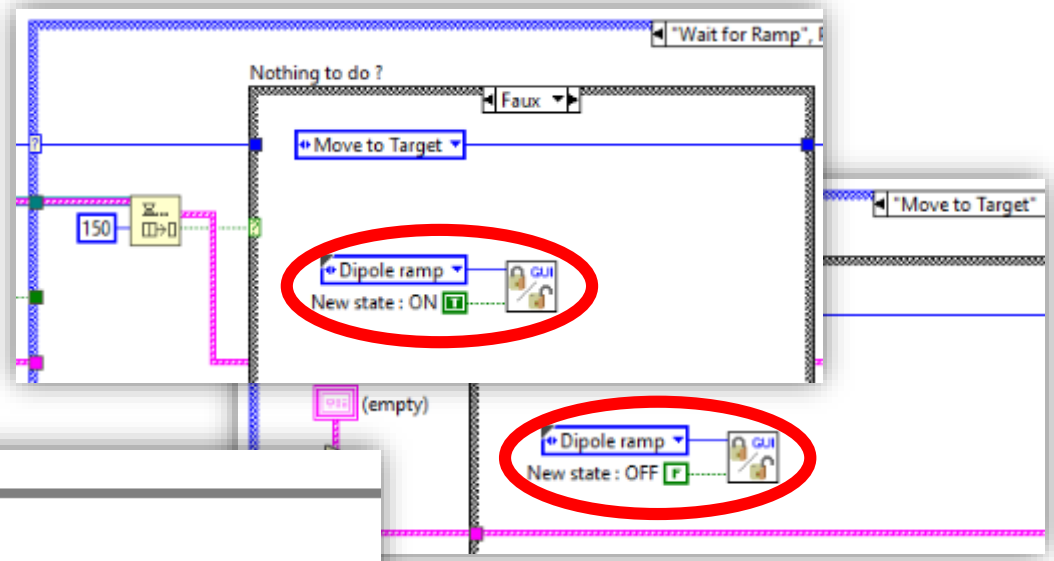
4.1 En (pré)-initialisation, câblez vers "Initialize.vi" les références (constantes) des commandes et indicateurs cibles.

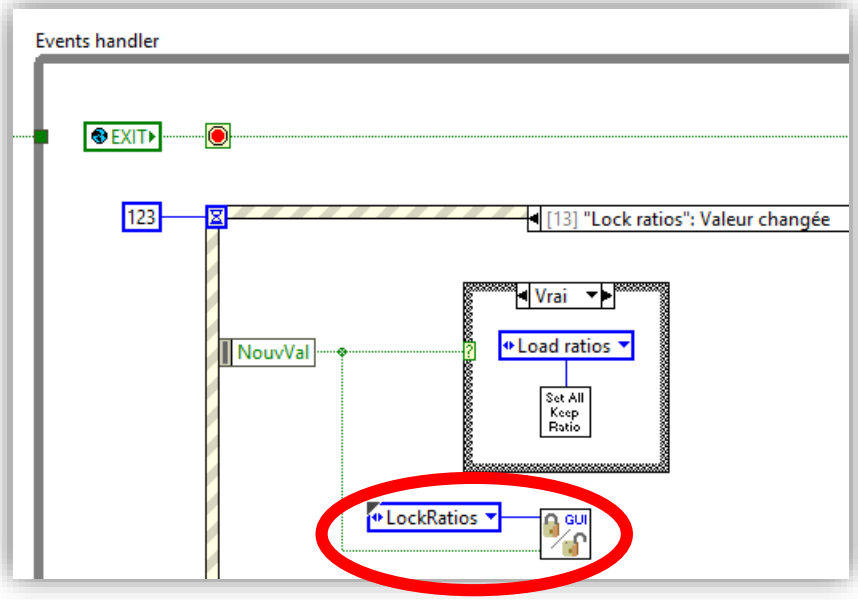


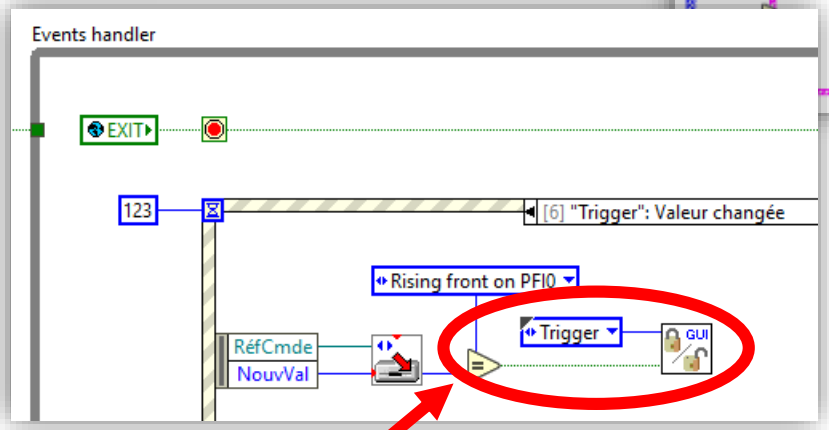
Note : Les états d'entrée sont présumés FAUX au démarrage. Si ce n'est pas le cas, il faut signaler les états VRAIS au moyen d'appels à "GUIInterlocks.vi" positionnés après "Initialize.vi".

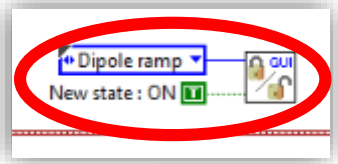
4.2 A travers le code : partout où un état d'entrée change de valeur, ajoutez un appel à "GUIInterlocks.vi" en spécifiant son nom et sa nouvelle valeur.

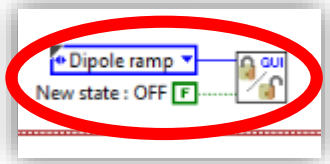












Note : On effectue ici les calculs éventuellement requis pour obtenir une information booléenne à partir des informations du contexte.

Conclusion : forces et faiblesses

- ⊕ Un seul VI simple assure la gestion des états et leur mise à jour (code léger).
- ⊕ L'application des états de sortie est immédiate (performance).
- ⊕ Les états de sortie sont tous recalculés à chaque changement d'un état d'entrée et uniquement à ce moment-là (garantie de cohérence, même en présence d'états interdépendants).
- ⊕ Un état de sortie n'est appliqué que s'il a changé (charge minimale).
- ⊕ La règle logique qui gouverne un état de sortie est centralisée, lisible (évolutivité, vérifiabilité).
- ⊕ L'intersection avec le code appelant est minimisée (découplage des facettes logicielles).
- ⊕ Les structures de code mise en œuvres sont plutôt simples (simplicité d'usage).

- ⊖ Nombreuses étapes manuelles pour configurer le code modèle (procédure nécessaire).
- ⊖ Les états d'entrée sont des abstractions : leur pertinence dépend du programmeur.
- ⊖ Recours à des variables globales pour passer les références à "GUIInterlocks.vi".
- ⊖ Mise en œuvre d'astuces peu conventionnelles (VI incorporé pour initialiser les globales, mémorisation des états d'entrée avec des indicateurs, usage de variables locales…).