De la modélisation par intension du problème de commande logique à la génération de code ST (IEC 61131-3)

Roisin Mathieu[1], Dimitri Renard [1], David Annebicque[1], Pierre-Alain Yvars[2] et Bernard Riera[1]
[1] CReSTIC, Université de Reims Champagne-Ardenne, Reims, France
[2] QUARTZ, EA 7393, Institut Supérieur de Mécanique de Paris (ISAE-Supméca), Saint Ouen, France
mathieu.roisin@univ-reims.fr

1 Introduction

L'arrivée de l'industrie 4.0 [1] bouleverse en profondeur la façon dont les systèmes industriels doivent être conçus et exploités. Face à la complexité grandissante des systèmes de production, il devient crucial de s'appuyer sur des méthodes solides, intégrées et automatisées, capables de prendre en compte, dès la conception, les différentes contraintes techniques, logiques et physiques. L'automatisation repose notamment sur la capacité à créer des contrôleurs logiques fiables et sûrs, adaptés aux besoins du système à piloter.

Dans ce cadre, notre travail s'intéresse à la résolution des problèmes de commande logique, et à la génération automatique de code pour les automates programmables industriels (API), en respectant la norme IEC 61131-3. Nous nous intéressons plus particulièrement dans cet article aux approches dites par intension, qui contrairement aux approches classiques par extension (qui représentent de manière explicite une solution comme c'est le cas avec le grafcet [2] ou les réseaux de Petri [3]), consiste à décrire les règles et/ou les contraintes qui définissent le problème permettant par la suite d'obtenir l'ensemble des solutions sans les énumérer toutes. Les deux approches ne sont pas incompatibles entre elles et doivent être combinées pour résoudre un problème de synthèse de contrôleur logique.

L'article est structuré de la manière suivante :

Après avoir présenté les deux approches (par extension et par intension) de résolution d'un problème, avec un point de vue sur la synthèse de contrôleurs logiques, un workflow existant dédié à la résolution des problèmes de synthèse de contrôleur combinant une approche par extension (grafcet) et une approche par intension à plat (la synthèse algébrique, SA [4]) est détaillé.

Cette démarche outillée permet de générer automatiquement du code ST conforme à la norme IEC 61131-3. Ce workflow sera ensuite appliqué à un cas d'étude (système de gestion de feux de

circulation) afin d'en identifier les limites, qui concerne principalement la définition des contraintes logiques de spécification.

Pour dépasser ces limites, nous proposons d'utiliser la méthode MBSS (Model-Based System Synthesis) [5], qui permet de construire un méta-modèle du problème de synthèse de contrôleur. Ce méta-modèle facilite la création de modèles génériques, hiérarchiques et instanciables, en s'appuyant sur le langage déclaratif DEPS [6], adaptés à la synthèse de contrôleurs logiques.

Enfin, nous présentons un nouveau workflow basé sur la méthode MBSS et le workflow initial du CReSTIC, qui permet également de générer du code ST (IEC 61131-3). Ce nouveau workflow est appliqué au même cas d'étude, et montre l'intérêt de l'approche.

2 Approche par extension ou par intension

Il existe dans la littérature deux méthodes (par extension ou par intension) pour résoudre un problème [7], quel qu'il soit. La différence entre résoudre un problème par extension ou par intension renvoie à deux manières de représenter et de traiter l'ensemble des solutions possibles d'un problème. L'approche par extension consiste à décrire explicitement une ou plusieurs solutions candidates. L'approche par intension consiste à décrire les règles et/ou les contraintes que doit satisfaire chaque solution du problème. La conception d'un contrôleur logique peut être vue comme un problème générique dont la résolution, peut se faire au moyen de méthodes par extension et/ou par intension.

La définition d'un contrôleur logique par une approche en extension, consiste le plus souvent à expliciter de manière formelle un candidat solution indiquant les états du système ainsi que les transitions possibles entre ces états. Cette représentation nécessite une vérification systématique de la conformité du comportement du système de commande candidat par rapport aux exigences fonctionnelles et de sécurité spécifiées. Le grafcet [2], les réseaux de Petri [3] ainsi que les automates à états [8] relèvent de cette approche très largement utilisée pour la commande des systèmes à évènements discrets. Cette méthode est aujourd'hui la plus répandue dans le monde industriel.

L'approche dite en intension de conception d'un contrôleur logique revient à exprimer les différentes exigences sous la forme de contraintes sur des variables représentant les inconnues du problème. Une ou plusieurs solutions admissibles peuvent alors être générées à l'aide d'un outil automatique de résolution qui garantit que chaque solution satisfait toutes les contraintes définies. Dans le domaine du génie automatique, des méthodes telles que la Supervisory Control Theory (SCT) Ramadge et Wonham [9] ou la synthèse algébrique (SA) [10] relèvent de l'approche en intension. Toutefois, ces méthodes restent encore peu utilisées dans le monde industriel, où elles sont souvent éclipsées par des approches en extension plus opérationnelles mais empiriques. On notera que l'un des avantages de la méthode de résolution par intension est l'obtention d'une solution nécessairement bonne par construction. En d'autres termes, si la spécification en intension est correcte, la solution le sera également. Enfin, on notera que les travaux sur la SA reposent sur une représentation à plat des contraintes de spécification. En d'autres termes, les contraintes sont définies au moyen d'opérateurs logiques (égalité, implication, ou, et, non) sur des variables. L'objectif est d'exprimer sous forme algébrique les variables inconnues (actionneurs par exemple) comme étant des fonctions logiques des variables connues (capteurs par exemple).

La synthèse algébrique (SA) [4] permet donc, à partir de la définition de bribes de spécification sous la forme de contraintes logiques de déterminer un ensemble de solutions algébriques satisfaisant nécessairement l'ensemble des contraintes spécifiées. Parmi ces solutions, il est ensuite possible d'en sélectionner une et de la convertir en code ST (Structured Text) [11].

La résolution d'un problème de commande doit pouvoir se faire de façon efficace en mixant les approches en extension et en intension. En effet, par exemple, la spécification d'un séquencement simple se fait facilement au moyen d'un grafcet (méthode en extension). En revanche, la prise en compte de la sécurité ou de la synchronisation de tâches au moyen d'une approche par intension présente a priori des avantages en termes de complexité. Une telle démarche a déjà été mise en place depuis plusieurs années. Il s'agit d'une proposition d'une démarche (Workflow) outillée, allant de la spécification (par extension et intension) à la génération automatique de code API [10], [12], [13].

3 Workflow pour la résolution d'un problème de synthèse de contrôleur

Le workflow (voir Figure 1) proposé permet de résoudre un problème de commande logique en mixant les approches en intension et en extension. Il repose sur 6 étapes :

1. La première étape consiste à définir les spécifications du contrôleur logique, en extension et/ou en intension, d'une partie opérative (PO) dont le modèle est, dans notre cas,réalisé avec FACTORY IO [12]. La spécification en extension se fait avec l'outil SFC-EDIT qui permet d'éditer des grafcets et de les exporter dans un format xml. La partie en intension repose sur l'écriture de contraintes logiques et d'un solveur algébrique [10].
2. À partir des contraintes logiques du problème, un modèle est construit au format BESS (Boolean Equation System Solver) [4].
3. L'outil BooG, qui a été développé en interne au laboratoire, utilise le solveur BESS, pour réaliser la synthèse algébrique du modèle. La solution obtenue est alors transformée en un grafcet (xml) qui va être combiné avec les grafcets partiels obtenus par extension afin d'obtenir le grafcet complet.
4. En utilisant le logiciel GReSTIC développé au laboratoire, le code ST du contrôleur logique est généré à partir de la scène FACTORY IO et du grafcet complet (xml).
5. L'outil GReSTIC intègre un automate logiciel permettant de simuler la Partie Commande et de se connecter à Factory IO. Cette mise en service virtuelle du système (PO+PC) permet à l'ingénieur de tester et de valider l'ensemble des éléments qu'il a définis au cours de ce processus.
6. Le code ST peut être enfin implémenté dans un API physique pour piloter l'installation industrielle.

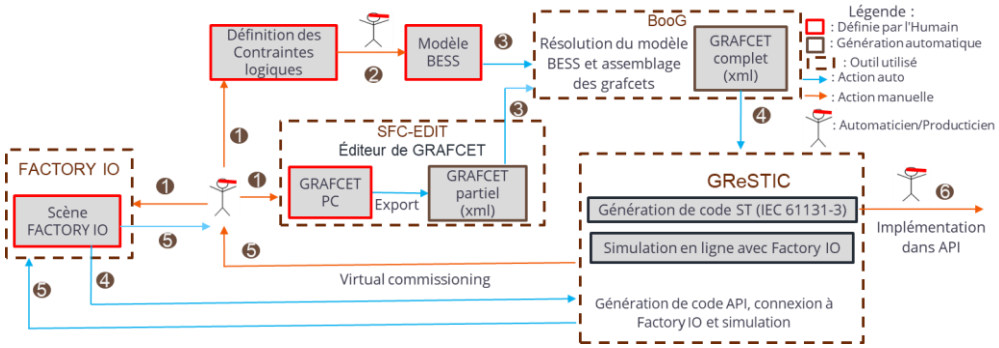

Figure 1 : Workflow de la génération de code ST avec la SA

4 Approche par intension par une modélisation à plat d'un problème de gestion de feux

Nous illustrons notre approche de la synthèse algébrique avec un système de croisement de deux convoyeurs (voir Figure 2 issu de l'école d'hivers SED). Chaque flux est régulé par un feu (rouge ou vert), et les colis doivent attendre le feu vert pour traverser la zone commune, rendue accessible par un plateau rotatif.

Deux capteurs indiquent la position du plateau

- TTLimite12 : Plateau en position 12
- TTLimite34 : Plateau en position 34

Les colis sont autonomes. En d'autres termes, ils s'arrêtent lorsque le feu est rouge et traverse lorsque le feu est vert. Ce fonctionnement est réalisé au moyen d'une commande spécifique (réalisée en grafcet, cf Figure 3) qui fait faire des allers-retours aux deux colis. Les signaux d'intention de traversée sont représentés par 4 étapes du grafcet (XR12, XR21, XR34, XR43).

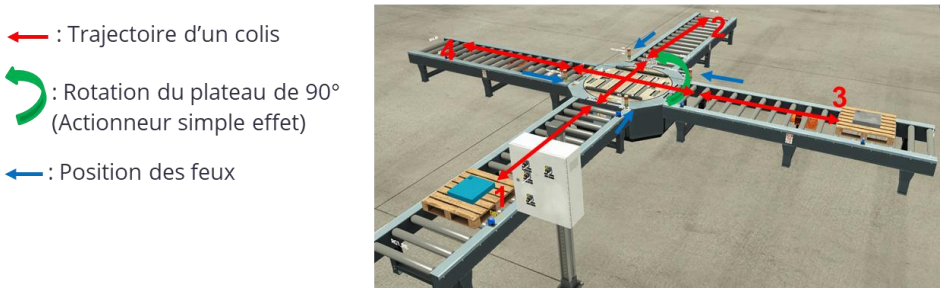

Figure 2 : Modélisation système du cas d'étude

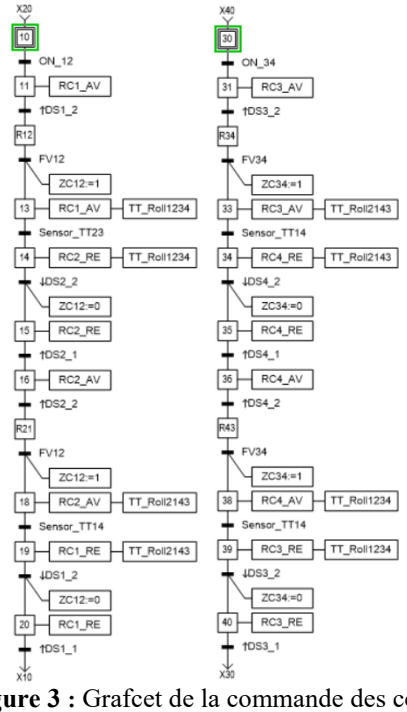

Figure 3 : Grafcet de la commande des colis

Deux variables globales (ZC12, ZC34) indiquent la présence d'un colis dans la zone commune. Deux boutons activables par l'utilisateur permettent de déterminer la position par défaut du plateau rotatif (ValDefaut) et la nécessité d'avoir le plateau qui se place dans la position par défaut (ActDefautMode). Neuf actionneurs doivent être commandés (feux verts/rouges et vérin du plateau) :

- Feu 12 : FV12/FR12, Feu 21 : FV21/FR21, Feu 34 : FV34/FR34 et Feu 43 : FV43/FR43
- TTTurn : Rotation du plateau rotatif simple effet (Actif = position 12, Inactif = position 34)

La commande consiste donc à synchroniser les feux et le plateau pour éviter les collisions, tout en assurant un retour automatique (ActDefautMode=1) ou non à la position par défaut (ValDefaut = 0 correspond à la position 34 du plateau tournant).

4.1 Modèle à plat du problème de commande

Ainsi le modèle du problème dans le formalisme de la SA se résume à une liste de contraintes correspondant aux exigences à satisfaire, une liste de contraintes correspondant aux hypothèses (entre des variables connues) et un critère de sélection de solution unique avec une maximisation des variables inconnues. Pour le problème du cas d'étude, nous avons défini quinze contraintes correspondant aux exigences, trois hypothèses et quatre maximisations lexicographiques (Figure 4)

```
(*----------------Exigence-----------------*)
FV12 = /(FR12);
FV34 = /(FR34);
FV12 <= (TTLimit12 . (AUT12 + ZC12));
FV34 <= (TTLimit34 . (AUT34 + ZC34));
(FV12 . FV34) = 0;
(ZC12 . /(FV12)) = 0;
(ZC34 . /(FV34)) = 0;
(ZC12 + AUT12) <= TTTurn;
(ZC34 + AUT34) <= /(TTTurn);
AUT12 <= (((XR12 + XR21) . /(ZC34)) . /(AUT34));
AUT34 <= (((XR34 + XR43) . /(ZC12)) . /(AUT12));
FV21 = FV12;FR21 = FR12;
FV43 = FV34;FR43 = FR34;
```

```
(*---------------------------- Hypothèse--------------------*)
(ZC12 . ZC34) = 0;
(TTLimit34 . TTLimit12) = 0;
(ZC12 . TTLimit34) + (ZC12 . /(TTLimit12)) + (ZC34 . /(TTLimit34)) + (ZC34 . TTLimit12)) = 0;

(*---------Maximisation----------*)
M0 : Maximal (**)
AUT34;
M1 : Maximal (**)
AUT12;
M2 : Maximal (**)
FV12 + FV34;
M3 : Maximal (**)
TTTurn . Maximiser + /TTTurn . /Maximiser;
```

Figure 4 : Extraction du modèle format BESS pour le problème du cas d'étude

Après résolution par BESS nous obtenons la solution suivante en Figure 5.

```
[One]      TT_Turn = ZC12+/ZC34.XR12./XR34./XR43+/ZC34./XR34./XR43.XR21+/ZC34./XR34./XR43.Maximiser
[One]        FV12 = ZC12+TT_Limit12.XR12./XR34./XR43+TT_Limit12./XR34./XR43.XR21
[One]        FV34 = ZC34+TT_Limit34.XR34+TT_Limit34.XR43
[One]        FR12 = ZC34+TT_Limit34+/TT_Limit12+/ZC12.XR34+/ZC12.XR43+/ZC12./XR12./XR21
[One]        FR34 = ZC12+/TT_Limit34+TT_Limit12+/ZC34./XR34./XR43
[One]        FV21 = ZC12+TT_Limit12.XR12./XR34./XR43+TT_Limit12./XR34./XR43.XR21
[One]        FV43 = ZC34+TT_Limit34.XR34+TT_Limit34.XR43
[One]        FR21 = ZC34+TT_Limit34+/TT_Limit12+/ZC12.XR34+/ZC12.XR43+/ZC12./XR12./XR21
[One]        FR43 = ZC12+/TT_Limit34+TT_Limit12+/ZC34./XR34./XR43
[One]       AUT12 = ZC12.XR12+ZC12.XR21+/ZC34.XR12./XR34./XR43+/ZC34./XR34./XR43.XR21
[One]       AUT34 = /ZC12.XR34+/ZC12.XR43
```

Figure 5 : Solution de la Synthèse Algébrique

4.2 Limites de l'approche

Malgré la capacité de la SA à produire du code conforme, cette modélisation "à plat" présente plusieurs limites. Tout d'abord, cette approche ne propose aucune représentation explicite et structurée du problème de commande à résoudre. Elle se limite à une collection de variables et de contraintes booléennes, ce qui nuit à la lisibilité et à la compréhension du modèle, en particulier pour des systèmes complexes. Ensuite, le modèle ainsi construit est peu extensible et difficilement généralisable. En effet, les contraintes sont formulées de manière très spécifique au cahier des charges initial, si bien que la moindre modification des exigences impose une révision majeure de l'ensemble des contraintes. Cela met en évidence un manque de flexibilité inhérent à cette approche. Ces limitations soulèvent la nécessité de repenser la structuration de la modélisation pour les problèmes de synthèse de contrôleurs logiques.

5 Approche en intension par une modélisation structurée du problème

5.1 L'approche MBSS

L'approche MBSS (Model-Based System Synthesis) [5] repose sur plusieurs principes fondamentaux :

- Une modélisation structurée du problème de conception, qui se concentre sur la définition du besoin plutôt que sur la description d'un système candidat.
- La prise en compte simultanée d'exigences hétérogènes, qu'elles soient fonctionnelles, logiques ou physiques.
- L'utilisation des exigences comme contraintes pour réduire l'espace des solutions, orientant ainsi la synthèse vers les configurations valides.

La résolution du problème se fait selon une approche dite de satisfaction des exigences, par opposition à l'approche classique de validation et vérification. En effet, à chaque étape du processus, l'objectif est de générer des solutions directement conformes aux exigences spécifiées. Les solutions obtenues sont donc correctes par construction. Cette approche s'inscrit dans une logique de cycle en I (voir Figure 6), qui se distingue du cycle en V traditionnel, en privilégiant une synthèse directe des solutions plutôt qu'une phase de vérification a posteriori. Elle a été jusqu'à présent utilisée pour faire de la synthèse d'architecture de systèmes techniques. Nous proposons dans ce travail de l'appliquer à la synthèse de modèles comportementaux (synthèse de contrôleur logique).

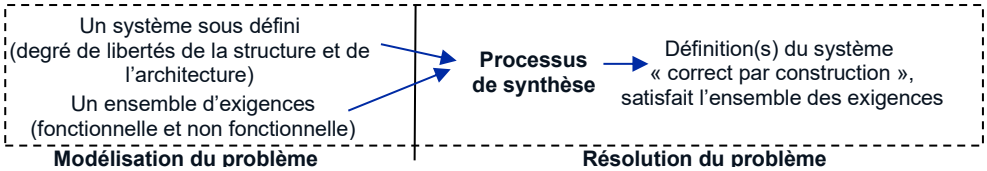

Figure 6 : Principe du MBSS

5.2 Le langage DEPS

Le langage associé à la méthode MBSS est le langage DEPS [6]. DEPS est un langage dédié à la modélisation de problèmes de conception de systèmes en vue de leur résolution automatisée. Il s'agit d'un langage textuel, déclaratif et structuré, basé sur la définition de modèles et de propriétés. DEPS permet de décrire une ontologie des grandeurs physiques et de représenter des variables ainsi que des propriétés dans des domaines discrets, continus et pseudo-booléens. En tant que langage déclaratif, DEPS permet à l'ingénieur de définir le problème de conception sans avoir à spécifier la manière dont il doit être résolu. La résolution est alors confiée à un outil dédié, équipé d'un solveur générique capable d'explorer l'espace des solutions de manière garantie. Chaque modèle DEPS peut être sous-défini et encapsule, dans un ordre précis, les éléments suivants : un ensemble d'arguments (Constants et/ou Elements), un ensemble de constantes (Constants), un ensemble de variables (Variables), un ensemble d'instances d'autres modèles (Elements) et un ensemble de propriétés à satisfaire (Properties). DEPS prend également en charge des relations d'agrégation, de composition et d'héritage entre modèles. Chaque modèle possède une signature, ce qui permet la surcharge et le polymorphisme, offrant ainsi un haut niveau de modularité et de réutilisabilité dans la modélisation.

5.3 L'IDE DEPS Studio

L'environnement intégré de modélisation et de résolution DEPS Studio, associé au langage DEPS, regroupe plusieurs composants clés : un éditeur de modèles, un gestionnaire de projet, un compilateur et un solveur. L'expérience a montré que la spécification d'un problème de conception de système est rarement correcte dès la première itération. L'expérience a montré que la spécification d'un problème de conception de système est rarement correcte dès la première itération. C'est pourquoi DEPS Studio intègre un solveur mixte (entiers, réels, pseudo-booléens) dédié, basé sur des techniques de programmation par contraintes, permettant à la recherche de solutions de participer activement au processus de validation et d'amélioration incrémentale du modèle. De nombreuses erreurs de modélisation ne peuvent être identifiées qu'au moment du calcul Un aspect fondamental de DEPS Studio est la préservation de la structure du modèle tout au long du processus, depuis la phase de modélisation jusqu'à la génération des modèles de calcul, garantissant ainsi la cohérence entre la spécification du problème et sa résolution.

5.4 Méta-modélisation structurée d'un problème de synthèse de contrôleur logique

Pour utiliser l'approche MBSS, il est nécessaire de définir les entités constitutives du problème ainsi que les interactions entre elles. Cela permet de formuler un ensemble d'exigences et d'identifier les degrés de liberté à explorer dans l'espace de conception. Dans le cadre de cet article, nous nous concentrons spécifiquement sur l'ordonnancement des tâches opératives à réaliser par le contrôleur d'un système manufacturier. Nous nous appuyons sur les caractéristiques définissant une tâche opérative, telles que décrites dans [14] et [15] :

- Les tâches opératives sont indépendantes les unes des autres.
- Une tâche, une fois déclenchée, n'interagit pas avec d'autres tâches et ne dépend d'aucune condition externe pour son exécution.
- Une tâche se termine dès que sa condition de fin est satisfaite.

La modélisation du problème de synthèse de contrôleur logique par intension requiert une formalisation explicite de l'ensemble des caractéristiques du système (système physique + contrôleur). Cela passe par la définition :

- D'un système sous-défini, c'est-à-dire comportant des degrés de liberté.
- D'un ensemble de contraintes (ou exigences) s'appliquant aux entités du système.

Nous proposons un extrait du méta-modèle (voir Figure 7) d'un problème de synthèse de contrôleur logique. Un tel problème est structuré autour de deux composantes principales :

1. Un système sous-défini, composé :
    - D'un système physique à contrôler.
    - D'un contrôleur sous-défini, dont le comportement est à synthétiser.
2. Un ensemble d'exigences, exprimées sous forme de contraintes à respecter.

Le système physique est modélisé à travers un ensemble de capteurs, possédant chacun un état courant représentant la valeur mesurée par le capteur.

Le contrôleur, quant à lui, est chargé de gérer l'exécution des tâches. Il est constitué :

- D'un ensemble de tâches opératives à ordonnancer.
- D'un ensemble d'observateurs, chargés de collecter des informations sur l'état des capteurs.

Chaque tâche correspond à une séquence d'actions que le contrôleur ordonne au système physique d'exécuter. Le comportement du contrôleur peut ainsi être vu comme le mécanisme d'autorisation de l'activation des tâches.

Les tâches peuvent être dans l'un des deux états suivants :

- Inactif : c'est à dire en attente d'activation.
- Actif : c'est à dire en cours d'exécution.

Le contrôleur est responsable de déclencher le passage de l'état « inactif » à l'état « actif ». Il est donc essentiel de connaître l'état précédent de chaque tâche pour pouvoir en déduire son état courant.

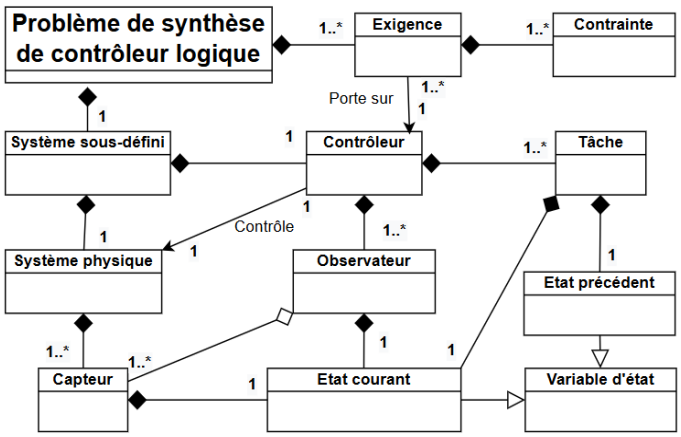

Figure 7 : Extrait du Méta-Modèle d'un problème de synthèse de contrôleur logique

Comme mentionné précédemment, une tâche peut se trouver dans l'un des deux états suivants : actif ou inactif. Le contrôleur doit donc respecter un ensemble d'exigences formulées sous forme de contraintes portant sur le comportement des tâches. Le comportement du contrôleur est défini par l'ensemble des tâches qu'il pilote et les contraintes associées à ces tâches.

Nous proposons de regrouper ces contraintes en trois catégories principales :

1. Contraintes liées aux observateurs
   Cette catégorie comprend un seul type de contrainte :
   - Contrainte de définition de l'observateur : elle permet de spécifier la relation entre l'état courant et l'état précédent d'une tâche, en fonction des mesures issues des capteurs constituant l'observateur. Elle appartient donc à l'observateur.
2. Contraintes inter-tâches
   Ces contraintes concernent les interactions entre différentes tâches :
   - Contrainte d'incompatibilité : elle interdit à certaines tâches d'être actives simultanément.
   - Contrainte de succession : elle impose qu'une ou plusieurs tâches dites suivantes ne puissent être exécutées qu'après la fin d'une ou plusieurs tâches dites précédentes.
3. Contraintes intra-tâche
   Ces contraintes sont associées à chaque tâche et définissent son comportement individuel :
   - Contrainte de condition initiale : elle impose que certaines valeurs d'état (issues des observateurs) soient satisfaites avant l'activation de la tâche.

- Contrainte de fin : elle définit les conditions, détectées par les observateurs, mettant fin à l'exécution de la tâche.
- Contrainte de comportement : elle décrit les relations internes entre les variables de la tâche et son état d'activité. Elle constitue un invariant propre à la tâche.

La Figure 8 présente une extension du méta-modèle illustrant cette taxonomie des contraintes. Les nouveaux éléments introduits dans le modèle y apparaissent en vert.

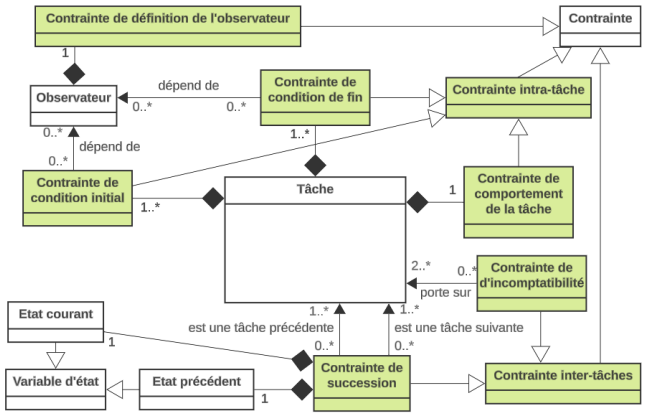

Figure 8 : Extension du Méta-Modèle avec les contraintes

5.5 Représentation du méta-modèle en DEPS

Un système sous-défini est ainsi composé de deux entités principales : un contrôleur et un système physique. La Figure 9, présente les modèles DEPS des tâches et des observateurs Le modèle de tâche est générique et peut être instancié pour représenter une tâche spécifique. Le modèle d'observateur, quant à lui, est abstrait : il doit être spécialisé pour chaque contrainte de définition d'observateur à l'aide d'un modèle dérivé.

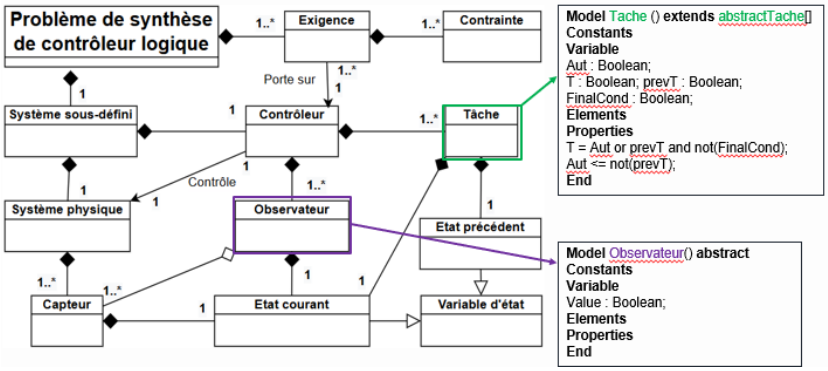

Figure 9 : Modèle générique de tâche et d'observateur correspondant au méta-modèle

De même, nous avons défini des modèles génériques de contraintes en DEPS (voir Figure 10), encapsulés dans le modèle d'exigence. Ces modèles peuvent être instanciés pour représenter un problème spécifique de synthèse de contrôleur logique.

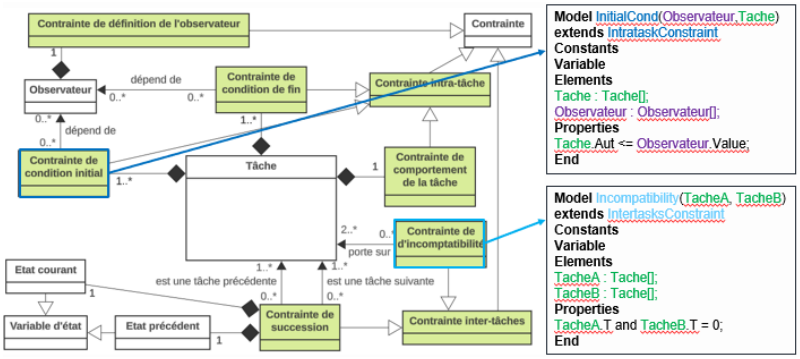

Figure 10 : Modèle générique de contraintes correspondant au méta-modèle

Ces modèles, de type « métier », offrent une représentation incrémentale du problème, facilitant son évolution. Par ailleurs, l'intégration d'un solveur dans l'environnement DEPS Studio permet de valider le modèle au fil de sa construction et de détecter précocement les erreurs ou l'absence de solutions, sans attendre la spécification complète.

6 Génération de code ST par modélisation structurée et hiérarchique par tâches

6.1 Méthodologie structurée pour la synthèse de contrôleur logique à base de tâches opératives

Cette partie présente une méthodologie générique pour représenter un problème de synthèse de contrôleur logique, en s'appuyant sur une modélisation structurée et hiérarchique par tâches opératives à ordonnancer. Le workflow détaillé de cette approche est illustré en neuf étapes clés :

1. **Définition des tâches et création des grafcets élémentaires pour chaque tâche**. À partir des spécifications du problème, on identifie les tâches opératives élémentaires. Il est important de choisir un niveau de granularité adapté : une tâche ne représente pas nécessairement un actionneur unique, mais peut regrouper plusieurs actions. Toutefois, des tâches trop globales risquent de sur-contraindre le système (absence de solution) ou de détourner la résolution du véritable problème. Chaque tâche doit ainsi rester simple et ne pas avoir de comportement conditionnel une fois activée. À ce stade, chaque tâche est modélisée par un grafcet simple.
2. **Identification des contraintes sur les tâches**. En s'appuyant sur les exigences et le comportement des tâches définies, on établit un ensemble de contraintes (inter-tâches, intra-tâche, et d'observateur), ainsi qu'un ordre de priorité si nécessaire.
3. **Modélisation du problème en DEPS.** En utilisant les modèles génériques (étape 0) proposés dans ce travail, on construit un modèle DEPS du système sous-défini (système physique et contrôleur) et des exigences. Ces modèles comprennent les entités suivantes : tâches, capteurs, observateurs et contraintes. Le modèle du problème est donc composé du système sous-défini et des exigences portant sur ce système sous-défini. Ce modèle DEPS peut ensuite être compilé sous DEPS Studio puis utilisé pour vérifier des propriétés ou bien réaliser des mises au point incrémentales grâce au solveur intégré de DEPS Studio.
4. **Création du grafcet de mémorisation des variables**. Étant donné que le fonctionnement d'un automate est séquentiel et cyclique, il est nécessaire de pouvoir accéder à l'état

précédent du système. Pour cela, un grafcet de mémorisation est ajouté afin de capturer l'état antérieur des variables critiques (associées aux tâches ou aux observateurs). Ce grafcet doit être positionné en amont des grafcets des tâches.

5. **Exportation du modèle au format BESS**. Une fois la modélisation validée dans DEPS Studio, un modèle BESS (à plat) est automatiquement généré. Son système d'équations regroupe les propriétés des modèles DEPS reliant l'état courant à l'état précédent.
6. **Assemblage du grafcet complet**. Le modèle complet comprend dans l'ordre :
    - Le grafcet de mémorisation des variables.
    - Les grafcets des tâches.
    - Le grafcet de la partie opérative (PO).

    L'ordre d'assemblage est crucial : le grafcet de mémorisation est placé en premier, suivi du grafcet des tâches, puis de celui de la PO.
7. **Synthèse et génération du code ST**. Le workflow synthèse algébrique (SA) (section 2.3) est ensuite utilisé sur le modèle BESS. Le code ST peut être généré automatiquement via GReSTIC. Ce code peut ensuite être utilisé pour simuler le comportement du système dans un environnement tel que Factory IO.
8. **Validation du contrôleur**. On simule le contrôleur logique avec GReSTIC et on pilote la scène Factory IO. On obtient donc une mise en service virtuelle du système. Permettant à l'ingénieur de valider l'ensemble des éléments qu'il a définis au cours de ce processus.
9. **Implémentation dans l'API**. Le code ST peut être implémenté dans un API pour piloter l'installation industrielle.

Trois étapes secondaires peuvent être réalisées après l'étape 3 :

A. **Choix des propriétés à respecter.** On choisit les propriétés qui doivent être vérifiées par les solutions.

B. **Vérification du respect des propriétés.** La résolution du modèle permet de trouver les états ne respectant pas les propriétés.

C. **Retour à l'étape 1 en cas de non-respect.** Dans le cas où la modélisation n'est pas suffisante pour le respect des propriétés souhaitées. Une modification des contraintes et/ou des tâches élémentaires est nécessaire.

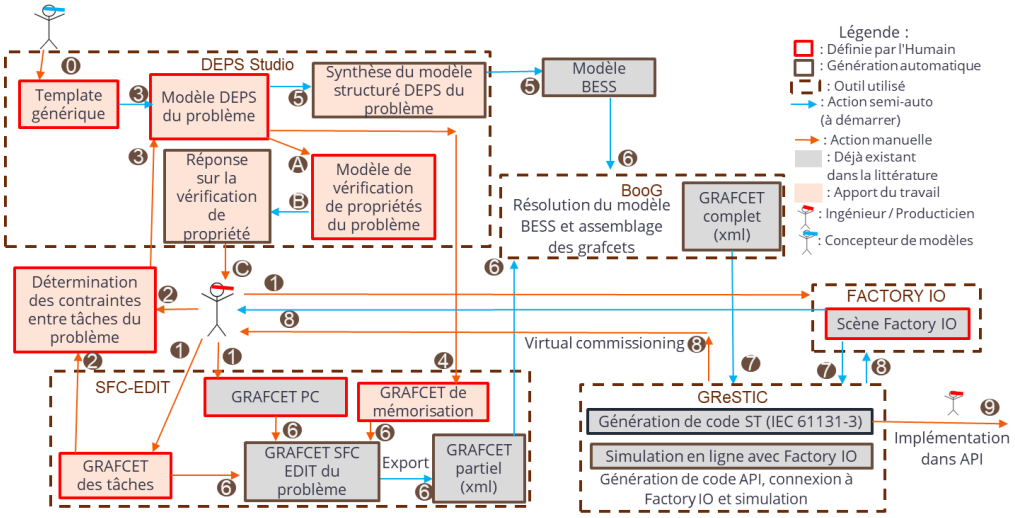

Figure 11 : Workflow de la génération de code ST avec notre méthodologie de modélisation structuré, hiérarchique et générique de problème de synthèse de contrôleur logique

6.2 Application sur le cas d'étude

Nous considérons dans cette partie le problème du cas d'étude de la section 4.

Dans un premier temps nous définissons les tâches constitutives du problème. Nous avons décidé de considérer trois tâches :

- Tâche « F12 » : Mettre en position le plateau rotatif pour la direction 12 puis allumer les feux verts 12 et 21 (éteindre les feux rouges 12 et 21). La tâche se termine lorsque le colis sort de la zone commune.
- Tâche « F34 » : Mettre en position le plateau rotatif pour la direction 34 puis allumer les feux verts 34 et 43 (éteindre les feux rouges 34 et 43). La tâche se termine lorsque le colis sort de la zone commune.
- Tâche « ParDefaut » : Mettre en position le plateau rotatif en position par défaut. Se termine lorsqu'un colis souhaite traverser la zone commune.

Dans un second temps, on identifie l'ensemble des exigences portant sur ces tâches. On peut ainsi identifier les contraintes suivantes portant sur ces tâches :

- L'incompatibilité entre toutes les tâches.
- Les conditions de fin pour chaque tâche sont les étapes de fin de chaque grafcet de ces tâches.
- Les conditions initiales :
    - Tâche « F12 » : Colis voulant traverser en 12 ou 21.
    - Tâche « F34 » : Colis voulant traverser en 34 ou 43.
    - Tâche « ParDefaut » : L'utilisateur souhaite une mise en position par défaut du plateau rotatif.
- Les priorités dans l'ordre suivant : « F12 » puis « F34 » puis « ParDefaut ».
  Dans la modélisation actuelle, les autorisations des tâches sont les seuls degrés de liberté. Leur maximisation garantit une solution unique et assure l'activation de chaque tâche dès que possible.

Dans un troisième temps, nous allons modéliser le problème en DEPS. Pour pouvoir créer le problème, il faut commencer par définir le modèle du système-physique qui est composé des différents capteurs définis dans le problème et des étapes de fin des grafcets de tâches. Le modèle du contrôleur est composé de tâches et d'observateurs. Le système sous-défini est composé de ces deux modèles (voir Figure 12).

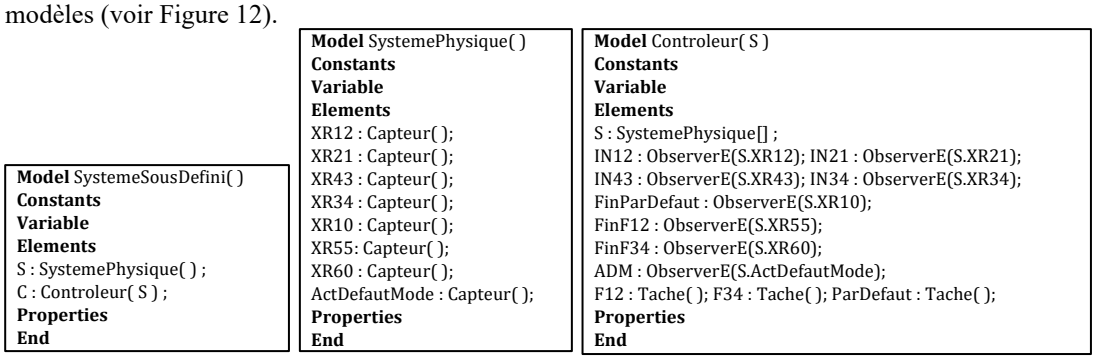

Figure 12 : Modèle DEPS du système physique, du contrôleur et du système sous défini du problème du cas d'étude.

Finalement, on peut exprimer les différentes contraintes qui s'exercent sur le contrôleur (Figure 13).

```
Model Exigence( C )
Constants
Variable
Elements
C : Controleur [SystemePhysique [ ] ] ;                          I12 : InitialCondition(C.Sys.IN12, C.Sys.IN21, C.F12);
I34 : InitialCondition(C.Sys.IN34, C.Sys.IN43, C.F34);           IParDefaut : InitialCondition(C.Sys.ADM, C.ParDefaut);
F12 : FinalCondition(C.Sys.FinF12, C.F12);                       F34 : FinalCondition(C.Sys.FinF34, C.F34);
FParDefaut : FinalCondition(C.Sys.FinParDefaut, C.ParDefaut);    IncompatibleAll : Incompatible(C.F12, C.F34, C.ParDefaut);
Properties
End
```

Figure 13 : Modèle des exigences du problème du cas d'étude en DEPS

Le problème (Figure 14) est composé du système sous-défini et des exigences portant sur contrôleur.

```
Problem DeuxTrains
Constants
Variable
Elements
Sys : SystemeSousDefini( ) ;          Exigence : Exigence(Sys.C) ;
Properties
Blo([max,max,max],[Sys.C.F12.aut, Sys.C.F34.Aut, Sys.C.ParDefaut.Aut]) ;
End
```

Figure 14 : Modèle du problème du cas d'étude en DEPS

Dans un quatrième temps, nous définissons un grafcet destiné à la mémorisation des variables. Ce grafcet est visible dans la vidéo qui illustre les cinq dernières étapes de la méthodologie. Ces étapes correspondent à une série de manipulations logicielles. La vidéo est accessible via le lien suivant : https://youtu.be/cEcR6Nl5QGc

La génération de code ST selon notre méthodologie permet d'obtenir une simulation pleinement opérationnel du cas d'étude. Cette approche structurée et hiérarchique repose sur des modèles « métiers » génériques et réutilisables, offrant à la fois une génération automatique de code et une simulation efficace du système. Par rapport aux méthodes classiques, elle fournit une représentation plus claire et intuitive, facilitant la compréhension. L'intégration d'un solveur garanti dans DEPS Studio permet en outre une mise au point efficace et une résolution précoce des modèles.

7 Conclusion et perspectives

Dans ce travail, nous avons proposé d'appliquer la méthode MBSS pour définir un méta-modèle du problème de synthèse d'un contrôleur logique chargé d'ordonnancer des tâches opératives. Pour cela, nous avons conçu des modèles génériques et hiérarchiques en DEPS, permettant de modéliser un problème générique de ce type. L'intégration d'un solveur de programmation par contraintes au sein de l'environnement DEPS Studio permet de garantir la correction et la cohérence des modèles et des solutions très en amont dans le processus de conception du système de commande. Ces modèles peuvent ensuite être réutilisés pour décrire divers cas de synthèse de contrôleur logique pilotant des tâches opératives. Cette modélisation s'inscrit dans un workflow permettant de générer automatiquement du code ST, en exploitant la structure hiérarchique et modulaire des tâches. L'ensemble de la démarche a été appliqué à un cas d'étude, présenté dans ce travail, où le modèle hiérarchique a permis de produire un code ST conforme aux exigences, ouvrant la voie à la simulation ou à l'intégration sur un système réel.

Nous envisageons de poursuivre ce travail en renforçant la généricité et l'expressivité du méta-modèle. Pour cela, nous explorerons différents cas d'étude afin d'identifier les limites de notre approche. Certaines contraintes spécifiques, non encore prises en compte, pourraient nécessiter l'ajout de nouveaux éléments au méta-modèle. L'une des principales limitations actuelles de notre workflow

réside dans le recours à la synthèse algébrique, qui impose que toutes les contraintes internes aux modèles soient exprimées uniquement dans le domaine booléen. Dans le futur, l'idée serait d'utiliser les capacités de résolution du solveur de DEPS Studio pour piloter directement la simulation.

Références

[1] Y. Koren, The Global Manufacturing Revolution: Product-Process-Business Integration and Reconfigurable Systems, John Wiley & Sons, 2010 Novembre.

[2] F. Schumacher, «Tool support for an automatic transformation of GRAFCET specifications into IEC 61131-3 control code,» *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA),* 2013, Septembre.

[3] S.-H. Teng, «Cellular manufacturing systems modeling: The Petri net approach,» *Journal of Manufacturing Systems,* pp. 45-54, 1990, Janvier.

[4] T. Ranger, «Approche par synthèse algébrique et filtre logique pour la commande des systèmes manufacturiers cyber-physiques,» *Université de Reims Champagne-Ardenne, Reims,* 2022.

[5] P.-A. Yvars, «Towards a correct by construction design of complex systems: The MBSS approach,» *Procedia CIRP,* pp. 269-274, 2022, Janvier.

[6] P.-A. Yvars, «DEPS: a model- and property-based language for system synthesis problems,» *Software and Systems Modeling,* pp. 973-1002, 2024, Août.

[7] J. Peregrin, «Extensional vs. Intensional Logic,» chez *Philosophy of Logic*, Elsevier, 2007, pp. 913-942.

[8] O. M. Rabin, «Finite Automata and Their Decision Problems,» *IBM Journal of Research and Development,* pp. 114-125, 1959.

[9] P. J. Ramadge, «Supervisory Control of a Class Of Discret Event Processes,» *SIAM Journal on Control and Optimization,* pp. 206-230, 1987.

[10] Y. Hietter, «Synthèse algébrique de lois de commande pour les systèmes à évènements discrets logiques,» *Université de Reims Champagne-Ardenne, Reims,* 2009.

[11] N. Jouvray, «Langages de programmation pour systèmes automatisés : norme CEI 61131-3,» *Techniques de l'Ingénieur,* 2008.

[12] T. Ranger, «Manufacturing Tasks Synchronization by Algebraic Synthesis,» *4th IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control CESCIT 2021,* pp. 226-231, 2021.

[13] D. Renard, «From Reinforcement Learning to Reality: Generating Structured Text Logic Controller,» *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT),* pp. 1269-1274, 2024.

[14] M. Roisin, «Constraint Programming for Logic controller Synthesis,» *10th International Conference on Control, Decision and Information Technologies (CoDIT),* 2024.

[15] M. Roisin, «Synthèse de système à base de modèles pour la conception de contrôleur logique correct par construction,» *19ème colloque national, Recherche et enseignement agiles pour une industrie soutenable (S.mart),* 2025, Mai.

[16] B. Riera, «HOME I/O and FACTORY I/O: a virtual house and a virtual plant for control education,» *20th IFAC World Congress,* pp. 9144-9149, 2017.