Modélisation et vérification d'un système mécatronique par SyncCharts

Daniel Gaffé

Laboratoire I3S (CNRS - UNSA)
Les Algorithmes - Bât Euclide B
2000 route des Lucioles – B.P. 121
F-06903 Sophia Antipolis Cedex
gaffe@i3s.unice.fr

RÉSUMÉ. *Les systèmes réactifs temps réel sont fréquents dans l'industrie. Pour les spécifier, nous avons opté pour une approche synchrone. Nous montrons sur une application concrète, l'intérêt d'une modélisation unique et homogène de la Partie Commande et de la Partie Opérative par* SYNCCHARTS.

ABSTRACT. *The real-time reactive systems are frequent in industry. To specify them, we have chossen a synchronous approch. We will show about a concrete application, the interest of an unique and homogeneous* SYNCCHARTS *model of the process control and the environment.*

MOTS-CLÉS : *Contrôle de Processus, Application réactive temps réel, Langage synchrone,* SYNCCHARTS, STATECHARTS, *Preuve.*

KEYWORDS: *Process control, real-time reactive application, synchronous language,* SYNCCHARTS, STATECHARTS, *proof.*

1. Introduction

Dans cet article, nous nous intéressons à la conception des systèmes de commande de processus industriels qui se rencontrent plus précisément dans l'industrie manufacturière. Pour cette classe d'applications, le temps, le séquencement et la concurrence d'actions revêtent une importance toute particulière. Parmi les approches de conception possibles, nous nous intéresserons plus particulièrement à l'approche synchrone par le biais du modèle SYNCCHARTS issu de la famille des STATECHARTS [HAR 87]. Dans une première partie, nous reviendrons sur les principaux constructeurs du modèle et nous préciserons pour chacun leur sémantique de comportement. Dans une seconde partie, nous traiterons un exemple d'application réelle (*Systèmes Automatisés de Production*) issu de la *mécatronique*. Cet exemple restera volontairement simple pour intégrer le plus d'aspects possibles : contrôle logique du processus, modélisation de l'environnement et de ses réactions face au contrôle, aspect temporel ... L'idée, est également de cerner tous les comportements souhaités *de l'application globale* en identifiant, puis en prouvant un ensemble probant de propriétés. C'est pourquoi, nous modéliserons d'abors la *Partie Commande* de l'application, puis nous intègrerons progressivement un modèle de la *Partie Opérative*.

2. SyncCharts : entre modèle d'état et paradigme synchrone

Nous caractérisons un système par ses changements d'état. Ces évolutions sont déclenchées par les changements des entrées. Chaque occurence d'événement (n'importe quel changement significatif dans l'environnement) détermine un *instant d'activation*. La simultanéité logique des événements prend alors un sens et conduit à un comportement spécifique du système. Chaque remise à jour de la prise en compte de l'environnement peut définir une commutation de temps logique. La sémantique des langages synchrones est basée sur l'hypothèse de préemption forte et assure que toutes les évolutions prennent un temps nul. En théorie, ceci implique que le temps d'exécution dure lui aussi un temps nul. Cette hypothèse forte, permet au concepteur de repousser le problème du temps physique au niveau implémentation. En pratique, cela signifie que le concepteur doit s'assurer que les réactions possibles du système seront terminées avant l'arrivée du prochain événement significatif.

En Europe, GRAFCET [IEC02, AFN82] a été le premier modèle concurrent qui a été qualifié de synchrone (au sens des langages synchrones) puisqu'il *synchronise* ses évolutions sur l'occurence des changements d'entrées [1] et que d'un point de vue externe ses évolutions sont instantanées. De nos jours, il est largement utilisé en contrôle de processus et dans l'industrie manufacturière pour spécifier et programmer des contrôleurs logiques.

1. Pour une forte proportion de la communauté des SED, c'est exactement la définition *d'asynchrone* ; le terme synchrone étant réservé aux systèmes évoluant sur horloge.

Dans la mouvance des langages synchrones textuels (ESTEREL, LUSTRE et SIGNAL [BOU 91, HAL 91, LEG 91]), sont également apparus des formalismes graphiques [MAR 90] dont SYNCCHARTS [AND 96] que nous allons développer maintenant. Il est basé sur la notion *d'atomicité des réactions* et la *diffusion instantanée* des signaux internes. Nous verrons sur l'exemple qu'il est capable d'exprimer facilement la décomposition hiérarchique de systèmes et convient bien pour implémenter des systèmes réactifs. Sa sémantique garantit une génération de code qui respecte les spécifications d'origine sur le comportement et permet la vérification symbolique de propriétés ("WYPIWYE") de G.Berry [2] [BER 89]).

SYNCCHARTS reprend dans son formalisme graphique tous les constructeurs ESTEREL de séquencement, de parallélisme, de préemptions et de suspension. Le compilateur SyncCharts génère d'ailleurs un code ESTEREL de comportement équivalent qui permet de bénéficier de tout l'environnement de développement d'Esterel. En conséquence, tout ceci a valu au modèle SyncCharts, le surnom *d'Esterel graphique*.

Avec l'opérateur de concurrence (ligne pointillée), SYNCCHARTS a de plus deux mécanismes particuliers d'exception : la préemption forte et la préemption faible. Considérons ainsi un exemple de régulation de chauffage.

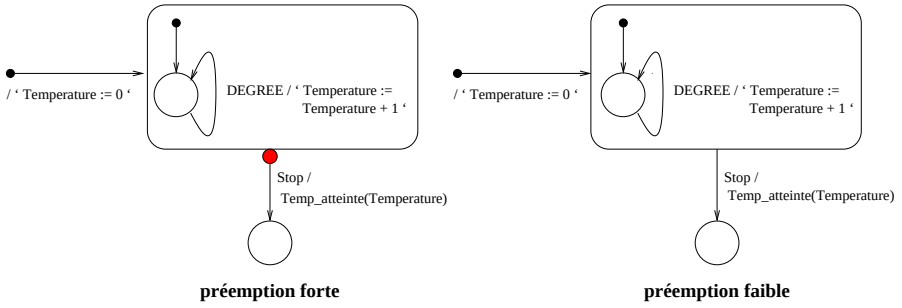

Figure 1. *Préemptions du modèle*

Sur la figure de gauche, la préemption (flèche avec rond) est forte : le signal `Stop` a priorité devant l'incrémentation de la variable et bloque celle-ci. Sur la figure de droite, quand le signal `Stop` arrive, la valeur de la température est émise après une possible incrémentation de `Temperature`.

En SYNCCHARTS, il existe également un opérateur de suspension qui permet de geler l'évolution de la partie de programme sur lequel porte le signal. Il se matérialise par une ligne droite terminée par un rond.

2. What You Prove Is What You Execute.

3. Traitement de l'exemple

3.1. *Objectif de l'application*

Le préhenseur pneumatique consiste à prendre un pignon dans un tiroir pour le monter sur un axe. Le système physique est constitué principalement par deux verins pneumatiques double-effet et d'une ventouse. Cet exemple est proposé comme banc d'essai (benchmark) par le groupe COSED (`www.lurpa.ens-cachan.fr/cosed`) du GDR Automatique pour expérimenter de nouvelles méthodes de conception et d'analyse des *Systèmes à événements discrets*.

La cinématique (cycle en U) du système est décrite par la figure 2. On remarque par exemple, que tous les déplacements horizontaux ne doivent se faire qu'en position haute.

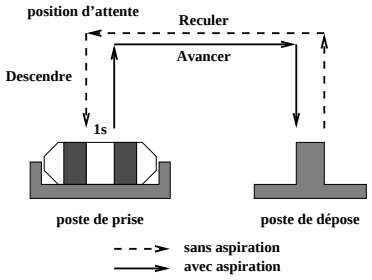

Figure 2. *Préhenseur pneumatique*

Le vérin du mouvement horizontal est piloté par un distributeur bi-stable nécessitant deux commandes (`Avancer` et `Reculer`). Le vérin du mouvement vertical est associé à un distributeur 5/2 mono-stable sensible à la commande `Descendre`. Cela sous-entend que l'absence de commande doit provoquer le retour du vérin dans sa position d'origine (ici position haute). La ventouse est elle-même activée par un distributeur mono-stable (l'aspiration se faisant par effet Venturi).

Dans la suite de cet article, nous allons nous intéresser à la *modélisation globale* de cette application : La méthode consiste à enrichir progressivement (ou à restreindre !) son comportement réel en partant de son modèle de commande *voulu par le concepteur* et en intégrant peu à peu ses aspects technologiques. Pour cet exemple en particulier, nous partirons de la commande logicielle, puis nous intègrerons la commande cablée, enfin nous intégrerons un modèle de comportement des pré-actionneurs et actionneurs (vérins, ventouse et distributeurs). Chaque niveau sera modélisé en SYNCCHARTS. Il intègrera par *hiérarchie SyncCharts* les niveaux précédents et fera l'objet d'une validation formelle spécifique.

Notons que l'article ne montre que le résultat final de cette analyse et cache (par manque de place) sa *dynamique*. En effet nous avons souvent été amené à reconsidérer les niveaux inférieurs (commande logicielle en particulier) à chaque fois qu'un des niveaux supérieurs ne donnait pas satisfaction.

3.2. *Modélisation de la commande exercée par l'automate programmable*

Le logiciel de commande est implanté sur un automate programmable industriel (API). Cet automate s'appuie sur la connaissance qu'il a de l'environnement (ici les capteurs de fins de course des vérins) pour générer la séquence des actions. Tous les signaux qui en résultent, sont regroupés sur la figure 3.

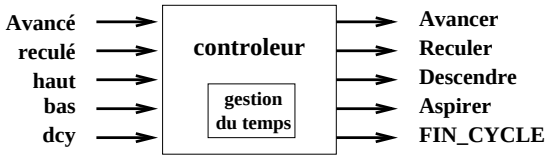

Figure 3. *Entrées-sorties nécessaires à l'API*

Le langage de programmation de l'API ne nous intéresse pas directement ici. Seul compte son modèle de comportement qui peut nous permettre de vérifier des propriétés de comportement. Au moins en Europe, beaucoup d'industriels ont pris l'habitude de modéliser leurs applications en GRAFCET. Dès 1980, des travaux de recherche ont été effectués pour prouver ce modèle [MOA 81]. Mais la vague principale a plutôt portée sur la période 1995 [LEP 94, ROU 94, GAF 96, DEL 96, LHE 97, LAM 98]. Ici, nous lui préfèrerons SYNCCHARTS pour deux raisons principales : D'abord la notion de hiérarchie (modes de marche et d'arrêt par exemple) est plus agréable à exprimer en SYNCCHARTS. Ensuite, nous allons modéliser l'environnement de l'API et il est toujours souhaitable d'homogénéiser le formalisme de représentation.

La figure 4 montre le niveau hiérarchique le plus élevé du contrôleur : La partie droite du syncCharts assure la gestion de la temporisation (une seconde) nécessaire à la préhension de la pièce par la ventouse. La partie gauche s'exécute de manière concurrente. Elle montre le séquencement de deux modes de marche :

– l'initialisation, souvent appelée "*mise en situation initiale de la partie opérative*

– le cycle normal

Enfin, si nous regardons plus en détail, nous remarquons la présence de la transition "triangle" SyncCharts qui indique que le cycle principal démarre dès que la phase d'initialisation est terminée.

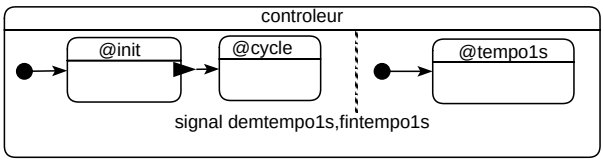

Figure 4. *SyncCharts global de contrôle*

Les *modes de marche* s'expriment eux-mêmes en SYNCCHARTS et peuvent à leur tour se décomposer : Sur la figure 5 en bas à droite par exemple, le transport réel de la pièce (ventouse active) a été isolé du reste du cycle.

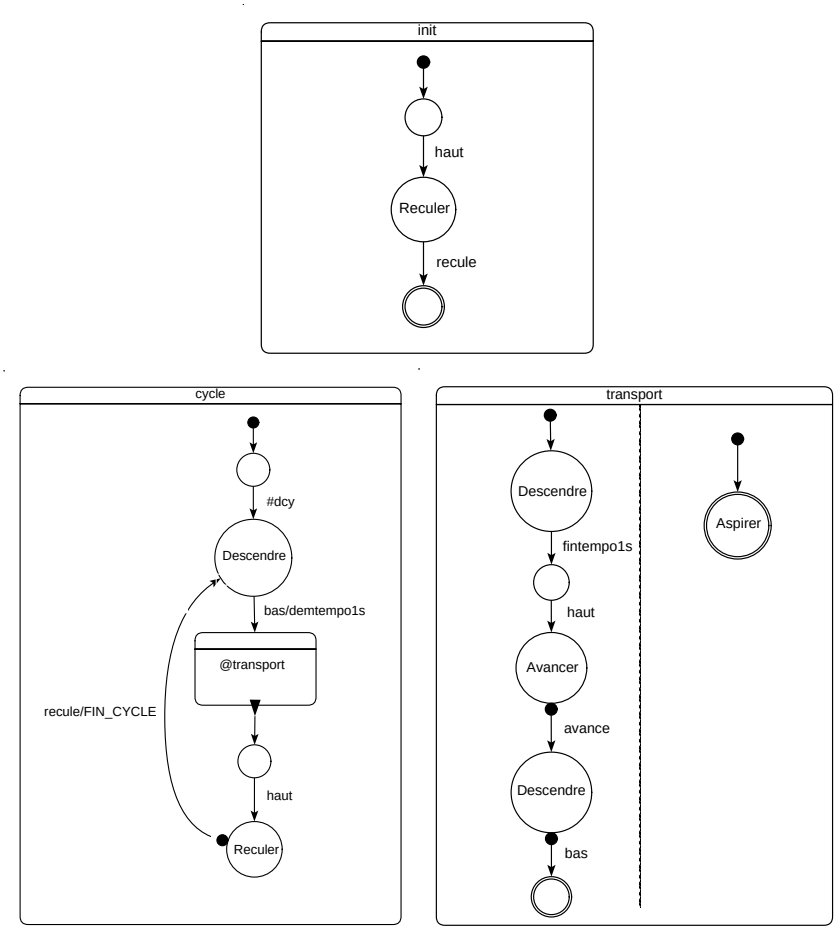

Figure 5. *Phase d'initialisation et de cycle normal*

Une fois modélisée par SYNCCHARTS, l'application peut bénéficier de l'environnement de développement d'Esterel, dont :

– le simulateur `xes` qui permet de tester des scénarii de séquences d'entrées,

– le prouveur symbolique sur équations booléennes d'états XEVE [BOU 97]

En particulier, XEVE a été utilisée ici pour montrer symboliquement que les ordres opposés `Avancer` et `Reculer` n'étaient jamais émis en même temps. Notons tout de même que ce type de preuve n'est pas spécifique à XEVE : Comme la propriété est

purement combinatoire, elle doit être démontrable par la majorité des outils basés sur les BDDs [BRY 86].

Mais XEVE peut également vérifier des propriétés séquentielles : il suffit pour cela d'intégrer au modèle SYNCCHARTS, les séquences des événements qui conduisent à l'erreur. Ces séquences seront elles-mêmes exprimées en SYNCCHARTS sous forme d'automates "lignes". Ces automates particuliers, qui évoluent en fonction de l'environnement et des réponses du système, sont souvent appelés *Observateurs*[HAL 93b, HAL 93a]. De cette manière (figure 6), nous avons pu montrer que l'aspiration ne s'interrompait jamais entre l'instant où la pièce commence à monter et l'instant où le vérin termine la descente.

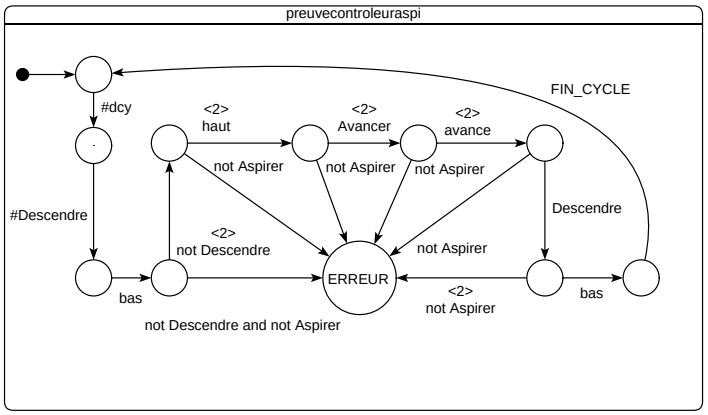

Figure 6. *Observateur de la commande* `Aspirer`

Enfin, XEVE permet de s'intéresser directement à des preuves de type "Accessibilité d'un état donné : A la question "le cycle termine-t'il ?", il a engendré une séquence suffisante des entrées qui conduit effectivement à la fin du cycle.

3.3. *Introduction d'un modèle de la commande câblée*

Avec la modélisation SYNCCHARTS du comportement de l'API, nous pouvons vérifier bon nombre de propriétés logiques combinatoires ou séquentielles. Mais ce niveau de modélisation est insuffisant pour une application réelle car l'automate programmable se trouve plongé dans un environnement réel susceptible de modifier, voire d'altérer son comportement. En fait, la bonne génération des commandes par l'automate, est une condition nécessaire mais pas suffisante.

Pour cette application, l'automate programmable est dépendant du plan de cablage de l'installation. Ici, un des circuits électriques (nommé "`energie`") est spécialement dédié à l'alimentation des sorties de l'automate. De ce fait, l'automate s'arrête com-

plètement en cas de rupture de son alimentation. Par contre, il reste actif et continue d'évoluer en cas de rupture du circuit "`énergie`".

Sur le système, pour des raisons de sécurité, le circuit "`énergie`" doit être amorcé (relais `KA1`) par un poussoir `mise en énergie` et peut être coupé n'importe quand par les poussoirs `arrêt d'urgence` ou `arrêt énergie`. Le syncChart fig. 7 montre ce couplage électrique : En cas de coupure d'énergie, le syncCharts `controleur` continue de vivre mais voit ses sorties inhibées ...

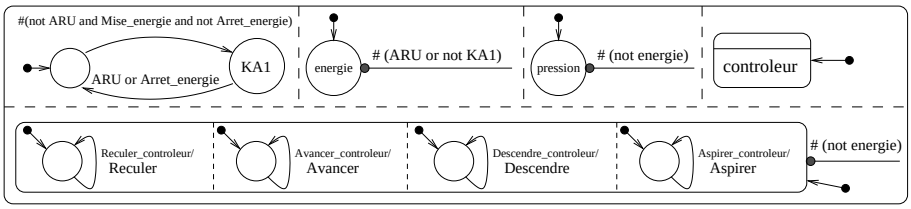

Figure 7. *Intégration du modèle de la partie cablée*

Notons également que l'application possède une sécurité pneumatique : La coupure d'énergie libère aussi un distributeur spécial mono-stable chargé d'alimenter en air tous les autres distributeurs et par voie de conséquence les verins. Le syncCharts précédent modélise cette propriété en préemptant le signal `pression` par `not energie`.

Finalement, nous pouvons considérer que le contrôle (Partie Commande) possède une partie cablée et une partie programmée. Les preuves de bon fonctionnement présentées au paragraphe précédent sont donc beaucoup plus probantes car elles intègrent maintenant ces deux aspects !

3.4. *Prise en compte de l'environnement du contrôleur dans les preuves de comportement*

Malheureusement, la modélisation de la Partie Commande (PC) ne suffit pas pour valider l'application complète. L'environnement (*la partie opérative* : PO) a lui-même un comportement propre et une mémoire. Sur notre application par exemple, rien ne peut prouver que la ventouse remonte effectivement dès que la commande `Descente` est coupée ...

Cet état de faits, nous pousse donc à modéliser également la PO et à intégrer le modèle obtenu dans les preuves de sureté. Dans cet article, nous nous appuyons sur l'expressivité de SyncCharts pour définir de multiples comportements locaux, sans avoir à l'avance une idée préconçue du comportement global de la Partie Opérative. Ainsi, nous donnons d'abord un modèle de comportement des vérins, puis un modèle de comportement des distributeurs et de leur couplage pneumatique avec les vérins.

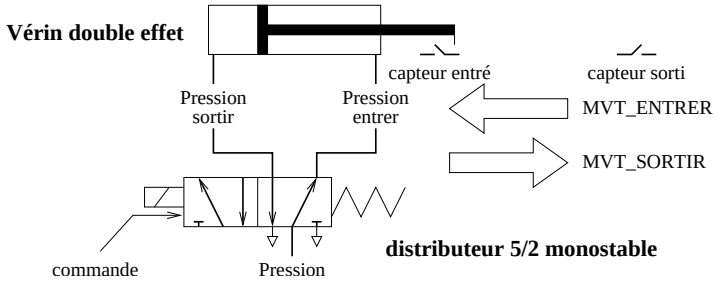

Figure 8. *Couple vérin/distributeur*

La figure 9 montre le modèle syncCharts choisi pour le vérin et son effet sur les capteurs de *fins de course*. La tige du vérin met un temps de 4 dixièmes de secondes [3] pour entrer ou sortir. Au départ, la tige est soit entrée, soit sortie soit à mi-course [4]. Remarquons de plus que l'environnement (où plutôt l'image que nous renvoie l'environnement) ne réagit pas instantanément à une sollicitation : l'état des *fins de course* est remis à jour dans un *instant logique différent* de la réaction proprement dite du vérin.

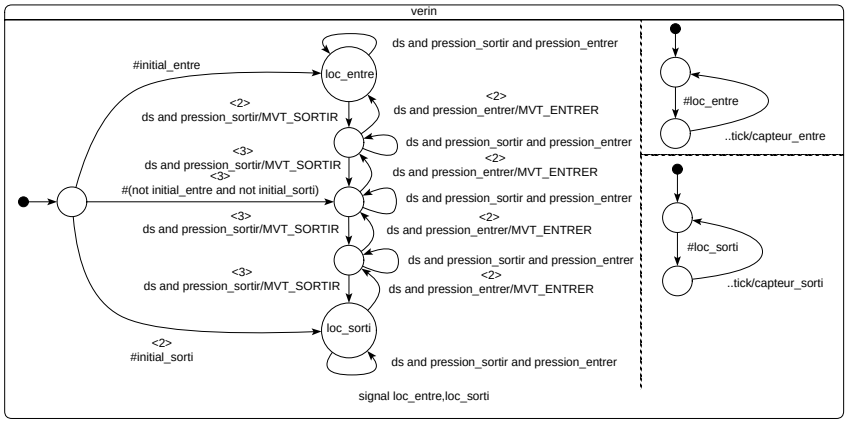

Figure 9. *Modèle du vérin (ds représente le dixième de seconde)*

Remarquons ici une limitation importante de l'approche synchrone : La prise en compte de l'environnement nécessite l'intégration explicite du temps. Dans notre approche, le temps est discrétisé et vu comme un évènement quelconque. Les langages synchrones n'apportent aucune aide méthodologique pour fixer la granularité temporelle nécessaire à l'élaboration de preuves de comportement probantes. Sur le modèle

3. Ce choix est bien sûr arbitraire ...
4. Cas possible lorsque aucune chambre n'est alimentée en pression.

du vérin, la granularité choisie est d'un dixième de seconde : Elle a conduit à un automate principal à cinq états (constitué des deux positions de fin de courses et de 3 positions intermédiaires). Une horloge plus fine aurait peut-être montré des phénomènes indésirables... Ceci rejoint le problème général de la finesse nécessaire qu'un modèle doit atteindre pour que la simulation ou les preuves aient un sens. En fait, il faut garder à l'esprit que : *On ne prouve que ce que l'on modélise ...*

Avec la même finalité, le tiroir du distributeur a été modélisé. Son déplacement et la présence (ou non) de pression en entrée, conditionnent la mise en pression du vérin double-effet. Ce déplacement ne peut pas être instantané et induit des effets transitoires. Dans le cas de l'arrêt (resp. apparition) de la commande, le distributeur passe par un état où la pression de retour n'est pas encore suffisante alors que la pression de sortie est encore présente. En conséquence, le vérin continue transitoirement son mouvement de sortie alors que la commande n'est plus présente ! Sur le syncCharts 3.4, ce phénomène a été modélisé par les deux transitions qui relient les macro-états.

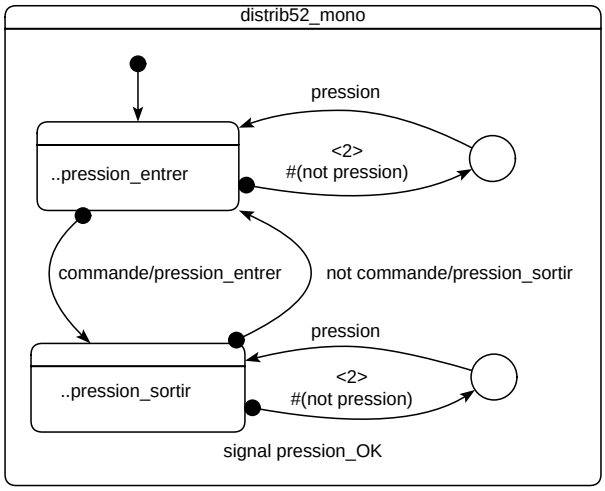

Figure 10. *Modélisation de l'action du tiroir sur la mise en pression du vérin*

Les différentes entitées spécifiées isolément peuvent maintenant être associées au sein d'un SYNCCHARTS unique qui intègre aussi bien la Partie Commande que la Partie Opérative. Ce SYNCCHARTS intègre également les propriétés combinatoires ou séquentielles que l'on souhaite vérifier.

Parmi ces propriétés, nous avons pu vérifier formellement (détection de 18461 états indépendants par XEVE) que tous les mouvements horizontaux ne s'effectent qu'en position haute dès l'ordre de `départ-cycle`. Sur la figure 3.4 associée à l'observateur, `MVT-AVANCER` et `MVT-RECULER` correspondent aux renommages respectifs des signaux `MVT-SORTIR` et `MVT-ENTRER` pour le verin horizontal. De plus, `haut` est

le renommage de `capteur-entré` pour le vérin vertical. Cet observateur s'appuie donc effectivement sur l'état des vérins pour tester la propriété et non plus sur leur commande ...

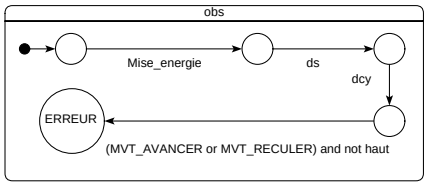

Figure 11. *Observateur des mouvements horizontaux*

XEVE a également démontré qu'un cycle complet prend un temps de 2.9 secondes (séquence de 29 ds) conformément aux modèles adoptés pour la Partie Opérative. Bien-sûr ces modèles et la granularité temporelle choisie ont une forte incidence sur la pertinence des preuves effectuées.

4. Conclusion

Dans cet article, nous avons montré l'intérêt de notre approche modulaire et hiérarchique basée sur SYNCCHARTS dans la conception des systèmes automatisés. La modélisation couvre aussi bien les aspects logiciels et matériels. (D'ailleurs SyncCharts peut également être utilisé en synthèse de la partie logicielle.) Disposer d'un modèle unique hiérarchique de comportement fut indispensable pour valider formellement chaque niveau avant de l'intègrer dans le niveau supérieur.

Avec les différents modèles de comportement présentés, des extensions sont bien-sûr envisageables. En premier, nous pourrions intégrer des pannes dans le système et s'intéresser aux comportements dégradés. Ceci induirait souvent des restrictions de comportements qui seraient naturellement modélisables par l'opérateur de suspension de SYNCCHARTS. Une autre voie est d'enrichir le comportement du système en incluant d'autres modes de marche comme la *marche manuelle sécurisée* ou des fonctionnements dégradés liés au pannes capteurs, L'aspect hiérarchique de SYNCCHARTS est là encore un atout pour représenter la commutation des modes, habituellement décrite en GEMMA.

Dans cet article, est également apparue une limitation sérieuse liée à l'approche synchrone elle-même et à la notion d'automate d'état fini : *la granularité du temps*. En fait, seul les automates temporisés et les réseaux de Petri temporisés sont susceptibles d'intégrer un temps continu. Des travaux sur le Grafcet ont d'ailleurs été faits dans ce sens [AND 98, LHE 97, AYG 93]. Mais les résultats obtenus sont difficilement généralisables et risquent de ne pas passer à l'échelle. Les outils associés à l'approche synchrone comme XEVE, sont au contraire capables de tester rapidement une propriété sur des automates conséquents.

5. Bibliographie

[AFN82] AFNOR, Paris (France), « Diagramme fonctionnel GRAFCET pour la description des systèmes logiques de commande », juin 1982, Norme Française, NF C 03 190.

[AND 96] ANDRÉ C., « SyncCharts : a Visual Representation of Reactive Behaviors », rapport n° RR 95–52, rev. RR (96–56), Rev. April 1996, I3S, Sophia-Antipolis, France.

[AND 98] ANDRÉ C., GAFFÉ D., ROBERT M., « Verifying Temporal Properties in GRAFCET », *Computational Engineering in Systems Applications (CESA)*, vol. 3, Hamammet (Tunisia), April 1-4 1998, IEEE-SMC, p. 38–43.

[AYG 93] AYGALINC P., DENAT J., « Validation of Functional GRAFCET Models and Performance Evaluation of the Associated System using Petri Nets », *APII*, vol. 27, n° 1, 1993, p. 81–93, Hermès.

[BER 89] BERRY G., « Real-time programming : General purpose or special-purpose languages. », *In G. Ritter, editor, Information Processing 89, Elsevier Science Publishers B.V. (North Holland)*, , 1989, p. 11-17.

[BOU 91] BOUSSINOT F., SIMONE R. D., « The Esterel language *Another look at Real-time Programming* », *Proceeding of the IEEE*, , n° 79, 1991, p. 1293–1304.

[BOU 97] BOUALI A., « XEVE : an ESTEREL verification environment », rapport, December 1997, CMA - ENSMP.

[BRY 86] BRYANT R. E., « Graph-based algorithms for boolean function manipulation », *IEEE transaction on Computers*, vol. C-35, n° 8, 1986, p. 677–691.

[DEL 96] DELOOR P., « Du TTM/RTTL pour la validation des systèmes commandés par Grafcet », PhD thesis, Université de Reims Champagne Ardenne, Octobre 1996.

[GAF 96] GAFFÉ D., « Le modèle GRAFCET : réflexion et intégration dans une plate-forme multiformalisme synchrone », PhD thesis, Université de Nice-Sophia Antipolis, Janvier 1996.

[HAL 91] HALBWACHS N., CASPI P., RAYMOND P., PILAUD D., « Programmation et Vérification des Systèmes Réactifs : Le Langage LUSTRE », *Technique et Science Informatique*, vol. 10, n° 2, 1991, p. 139–157.

[HAL 93a] HALBWACHS N., *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers, Amsterdam, 1993.

[HAL 93b] HALBWACHS N., LAGNIER F., RAYMOND P., « Synchronous Observers and the Verification of Reactive Systems », NIVAT M., RATTRAY C., RUS T., SCOLLO G., Eds., *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993, Workshops in Computing, Springer Verlag.

[HAR 87] HAREL D., « STATECHARTS : A visual formalism for complex systems », *Science of computer programming*, vol. 8, 1987, p. 231–274.

[IEC02] IEC, Genève (CH), « Grafcet specification language for sequential function charts », 2002, International standard IEC 60848.

[LAM 98] LAMPÉRIERE-COUFFIN S., « De la vérification de cahiers des charges de systèmes à événements discrets à la validation des spécifications décrites en Grafcet », PhD thesis, École Normale Supérieure de Cachan, Janvier 1998.

[LEG 91] LE GUERNIC P. and all, « Programming Real-Time Applications with SIGNAL », *Proceeding of the IEEE*, vol. 79, n° 9, 1991, p. 1321–1336.

[LEP 94] LEPARC P., « Apports de la méthodologie synchrone pour la définition et l'utilisation du langage Grafcet », PhD thesis, Université de Rennes 1, Janvier 1994.

[LHE 97] L'HER D., « Modélisation du Grafcet temporisé et vérification de propriétés temporelles », PhD thesis, Université de Rennes 1, Septembre 1997.

[MAR 90] MARANINCHI F., « ARGOS : un langage graphique pour la conception, la description et la validation des systèmes réactifs. », PhD thesis, Université Joseph Fourier, Grenoble I, Janvier 1990.

[MOA 81] MOALLA M., « Spécification et conception sure d'automatismes discrets complexes, basées sur l'utilisation du Grafcet et de réseaux de Petri », PhD thesis, INP Grenoble, 1981.

[ROU 94] ROUSSEL J. M., « Analyse de Grafcets par génération logique de l'automate équivalent », PhD thesis, École Normale Supérieure de Cachan, décembre 1994.