
Modèle d'architecture générique pour la supervision de systèmes de production

Servat David¹, Chiron Fabien², Kouiss Khalid², Meurisse Thomas³, Souchet Guy⁴

1 CEA-List DTISI/SOL-SCRI, 91191 Gif-sur-Yvette

2 LIMOS-IFMA, BP 265, 63175 Aubière

3 SINOVIA, 93 rue Henri Rochefort, 91000 Evry

4 NEWTEC, 85290 St Laurent-sur-Sèvre

Résumé

Le projet CLIPS propose de tirer parti des résultats de recherche en matière d'ingénierie dirigée par les modèles et du savoir-faire de partenaires de R&D en matière de plates-formes à base de composants pour définir une approche innovante de conception de systèmes de production automatisés. Après un bref aperçu du contexte du projet CLIPS, ce papier présente un certain nombre de points clés de la démarche méthodologique proposée. La dernière partie traite de la définition d'une taxonomie du matériel rencontré pour la réalisation du système et de la projection des modèles vers les plates-formes d'exécution cibles logicielle ainsi que matérielle.

1. Introduction

L'un des buts poursuivis dans le projet CLIPS supporté par le programme RNTL est de réunir des industriels spécialisés dans la conception de plates-formes à composants et des chercheurs travaillant dans le domaine de l'ingénierie des modèles, en vue de proposer une méthodologie incrémentale de conception adaptée au domaine des systèmes de production automatisés. Le contexte applicatif est donné par la réalisation d'un nouveau type de palettiseur mettant tirant parti de la collaboration de plusieurs robots et équipements industriels (cf. illustration figure 1). Les propriétés attendues sont une plus grande souplesse et adaptabilité aux contraintes du domaine en tirant parti des démarches de conception dirigée par les modèles et à base de composants [1, 2, 3, 4 5]. Le domaine de la conception des systèmes de production est en effet en plein essor à l'heure actuelle mais n'offre pour le moment que peu de tentatives d'utilisation de ces démarches issues du génie logiciel.

Après un bref aperçu du contexte du projet CLIPS, ce papier présente un certain nombre de points clés de la démarche méthodologique proposée. La dernière partie traite de la définition d'une taxonomie du matériel rencontré pour la réalisation du système et de la projection vers les plates-formes d'exécution cibles logicielle ainsi que matérielle.

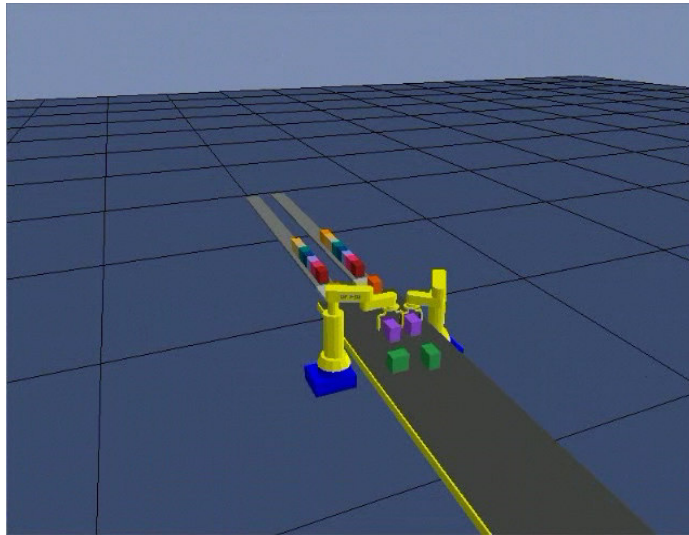


Figure 1: contexte applicatif

2. Ingénierie dirigée par les modèles

Le passage d'une approche centrée sur le code et la programmation vers une approche centrée sur la modélisation de systèmes vise à mieux faire face à la complexité accrue des systèmes industriels. L'Object Management Group (OMG) a lancé une initiative de promotion de cette démarche à travers son architecture dirigée par les modèles - Model Driven Architecture (MDA) [6]. Elle repose sur un processus de conception itératif de modèles UML. Cette démarche a suscité de nombreuses questions de recherche parmi lesquelles celle du développement d'un environnement de conception dirigée par les modèles est l'une des plus critiques.

2.1 Evolutions récentes de UML2 pour la conception à base de composants

La nouvelle version du langage UML apporte des avancées significatives à la modélisation de systèmes à base de composants, qu'ils soient logiciels ou bien matériels, en introduisant de nouveaux concepts hérités de différents domaines de recherche, depuis la conception orientée composants de plates-formes logicielles à la modélisation d'architecture de systèmes [3, 4, 5]. Un composant peut être ainsi modélisé à tous les niveaux du cycle de développement et successivement raffiné lors du déploiement et de l'exécution. Lors de ces phases, un composant s'exécute dans le système (on parle de manifestation ou "manifest") sous la forme d'un ou plusieurs artefacts, eux-mêmes déployés dans un environnement d'exécution.

La structure interne du composant est cachée et accessible uniquement à travers les interfaces fournies. Les interfaces requises modélisent les dépendances externes. Un modèle de composant peut également comporter une vue interne (ou boîte blanche) sous la forme d'un ensemble de classifiées réalisant le comportement du composant (diagramme de structure composite).

Cette vue en boîte blanche montre les relations entre les différents éléments et les connecteurs qui les relient. Ces connecteurs décrivent les patterns de communication qui sont valides dans ce contexte particulier. Si l'on sort de cette vue interne, on obtient une vue externe ou boîte noire.

Chaque partie contenue dans le composant peut être considérée comme un ensemble de classes assemblées d'une manière particulière dans le contexte du composant. Cette vision est héritée du domaine des ADLs (Architecture Description Languages) pour lequel existe depuis longtemps un besoin de spécifier des configurations particulières de composants et des détails dans la structure interne de chaque composant.

2.2 Besoins en méthodologie

Le modèle de composant UML2 et les concepts généraux ne suffisent pas pour décrire les principales facettes des systèmes embarqués temps réel. Une solution consiste à étendre ce modèle à l'aide d'un profil dédié, dans lequel un certain nombre de concepts facilitent la description non ambiguë d'exigences.

Ainsi de nombreux projets de recherche sont menés dans ce sens. D'une manière générale on peut identifier trois grands axes de recherche :

- les capacités de description cherchant l'exhaustivité ou spécialisation du langage pour permettre la modélisation d'aspects propres au domaine (matériel, temps physique, qualité de service, etc.).
- les possibilités de vérification de la cohérence d'un assemblage: la norme laisse entière liberté quant à la définition de règles de bonne composition entre composants exposant leurs services. L'incorporation de machines à état pour définir des protocoles d'utilisation des services fournis par des interfaces n'est pas non plus bien définie. Cela recouvre également l'aspect sémantique des modèles.
- des propositions méthodologiques pour utiliser l'ensemble de ces concepts qui paraissent parfois complexes et redondants (cf. débat sur le concept de port au sein de la communauté UML).

Ces préoccupations se retrouvent dans la démarche de spécification de SysML (System Modeling Language) qui vise à définir sur la base du standard UML un langage de modélisation de système supportant l'ensemble des étapes de spécification, d'analyse, de conception, de vérification et de validation pour une large gamme de systèmes complexes (logiciel, matériel, gestion de données, processus industriels, etc.). Actuellement SysML s'organise sous la forme d'une RFP (Request for Proposal) de l'OMG autour de UML pour l'ingénierie des systèmes (UML for SE RFP; OMG document ad/05-01-03). Une première spécification complète devrait être achevée au cours de l'année 2005.

Depuis plusieurs années, le CEA List a été impliqué dans ce champ de recherche appliqué à la conception de systèmes temps réel embarqués. Ces travaux ont abouti à la définition d'une plateforme de conception Accord/UML [7, 8, 9] fournissant une méthodologie et un ensemble d'outils d'assistance à la conception.

L'expérience accumulée dans ces travaux a motivé l'adaptation de cette démarche méthodologique au domaine des systèmes de production automatisés dans le cadre du projet CLIPS.

3. Application aux systèmes de production

Dans cette partie nous présentons quelques uns des principaux points méthodologiques mis en oeuvre dans le projet CLIPS [13].

3.1 Décomposition fonctionnelle

La description d'une décomposition fonctionnelle d'un système ou sous-système en composants s'appuie sur trois packages :

- un package **Actors**
- un package **UseCases**
- un package **Components**

2. par convention les acteurs placés à gauche sont les acteurs "actifs" (utilisateurs des fonctionnalités), ceux placés à droite sont "passifs" (fournisseurs de services complémentaires). Cette convention se retrouvera dans les diagrammes en boîte noire des composants - interfaces fournies et requises respectivement à gauche et à droite (cf. plus loin).
3. par ailleurs on représente l'ensemble des acteurs impliqués en associant seulement les acteurs parents aux cas d'utilisation.
4. si l'on requiert des fonctionnalités définies à l'extérieur du cluster en cours de modélisation, faire figurer les composants sous forme d'acteurs stéréotypés "component" (au besoin ajouter ce stéréotype au modèle).

Package Components

But: les fonctionnalités décrites dans le package UseCases sont amenées à être prises en charge par un ou plusieurs composants, constituant la structure interne du système considéré. Cette structuration et cette décomposition sont décrites dans le package Components.

Contenu:

- un composant auquel est attaché un diagramme d'activité appelé Scenario
- un ensemble d'interfaces regroupant des opérations
- deux diagrammes de déploiement : BlackBoxView et WhiteBoxView
- un diagramme de classe CompositeView
- un ensemble de packages : un par sous-composant formant la structure interne du composant considéré

Ce package Components regroupe divers éléments. Dans cette section focalisée sur l'aspect décomposition fonctionnelle nous nous intéresserons avant tout au diagramme de classe appelé CompositeView. Les autres éléments décrivant la structure proprement dite du composant sont décrits dans les sections suivantes.

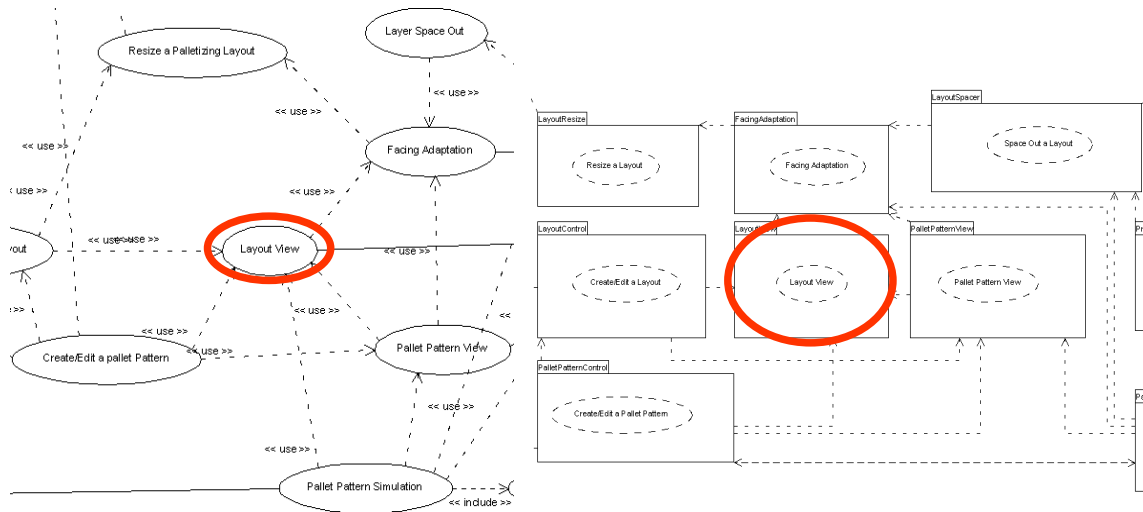


Figure 3: Diagramme CompositeView : définition fonctionnelle des composants

Le diagramme CompositeView représente la structure interne du composant considéré du point de vue décomposition fonctionnelle. Il montre comment les fonctionnalités décrites dans UseCases sont distribuées aux différents composants. Chaque package représente un composant de même niveau de décomposition et regroupe sous forme d'une ou plusieurs collaborations, l'ensemble des fonctionnalités qui ont été attribuées au composant considéré.

Règles de modélisation:

1. on figure l'attribution des cas d'utilisation aux différents composants par l'ajout d'une collaboration à l'intérieur du package représentant le composant.
2. à chaque cas d'utilisation du diagramme UseCases doit correspondre une collaboration portant même nom et attribuée à un unique composant. Ainsi une règle de bonne conception est de veiller que chacun des use cases décrits dans le package UseCases se retrouve bien sous la forme d'une collaboration dans une CompositeView à un niveau de récursivité quelconque.
3. les mêmes relations de dépendance existent entre cas d'utilisation d'une part et composants d'autre part.
4. cette description est appliquée récursivement lorsqu'un composant est lui-même décomposé de manière plus fine. Dans ce cas, il comportera également une CompositeView et un ensemble de packages représentant ses sous-composants, selon les mêmes principes définis ici.

3.2 Modèle des composants

Cette section décrit les éléments permettant de modéliser un composant proprement dit. Ces éléments se retrouvent dans le package portant le nom du composant. Nous ne reviendrons pas ici sur la décomposition fonctionnelle décrite par la CompositeView (cf. plus haut). Mais nous nous intéresserons à la modélisation structurelle du composant en deux vues principales :

- une vue en boîte noire décrivant les services fournis et requis par ce composant, découlant de l'attribution des collaborations
- une vue en boîte blanche qui décrit l'assemblage des composants internes le constituant

Cette approche de modélisation hérite des concepts définis dans UML2.0 (cf. discussion plus haut).

Vue BlackBox

But: ce diagramme montre un point de vue extérieur sur le composant considéré. On y trouve l'ensemble des opérations fournies par le composant regroupées au sein d'une ou plusieurs interfaces. On y trouve également la liste des interfaces requises par le composant considéré – ces interfaces requises étant des interfaces fournies par des composants externes. N'est pas détaillée sur cette vue la réalisation concrète des opérations fournies par le composant, ce qui sera l'objet de la vue en boîte blanche (cf. plus loin).

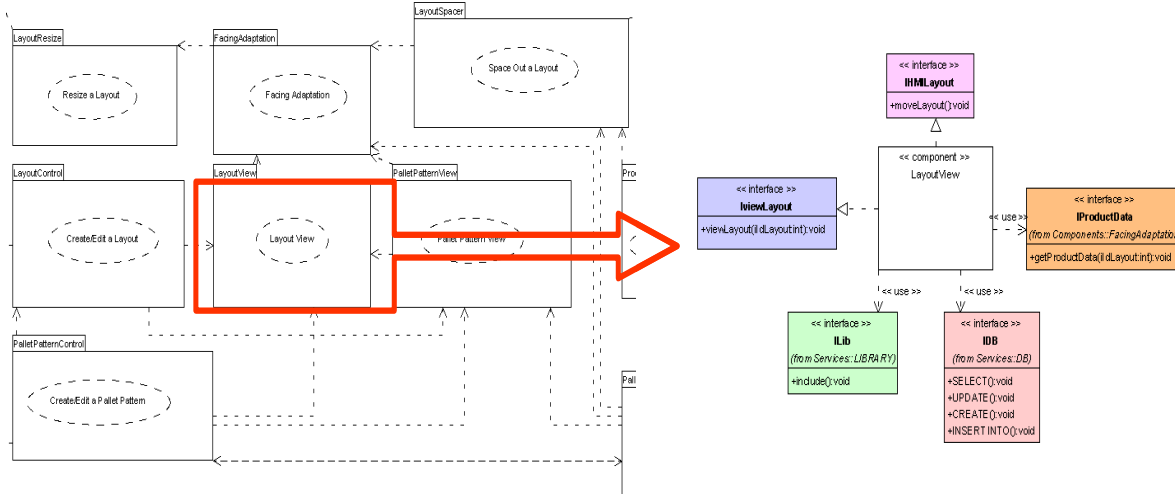


Figure 4: vue BlackBox

Règles de modélisation:

1. le lien entre interfaces fournies et composant est un lien d'implémentation
2. le lien entre interfaces requises et composant est un lien d'utilisation
3. autant que possible sans nuire à la lisibilité du diagramme, les interfaces fournies seront disposées à gauche du diagramme et les interfaces requises sur la partie droite (comme pour le diagramme de cas d'utilisation)
4. des conventions de noms et de couleurs peuvent être introduites pour faciliter la compréhension du diagramme : par exemple, pour distinguer les interfaces requises des interfaces fournies, parmi les interfaces fournies celles qui sont réalisées à l'intérieur du composant conteneur de celles réalisées par des composants externes), ou mettre en relief des interfaces particulières comme des IHM ou des accès à des bases de données – interfaces partagées par un large panel de composants et pouvant être regroupées sous forme de services du conteneur de composants.

Vue WhiteBox

But: ce diagramme montre la structure interne d'un composant composite. On y trouve l'assemblage des composants de grain plus fin constituant le composant considéré. Cette vue complète la vue précédente. Ici l'on voit clairement quels sont les composants de grain plus fin qui réalisent les opérations fournies par le composant les encapsulant – mentionnées dans le diagramme BlackBoxView.

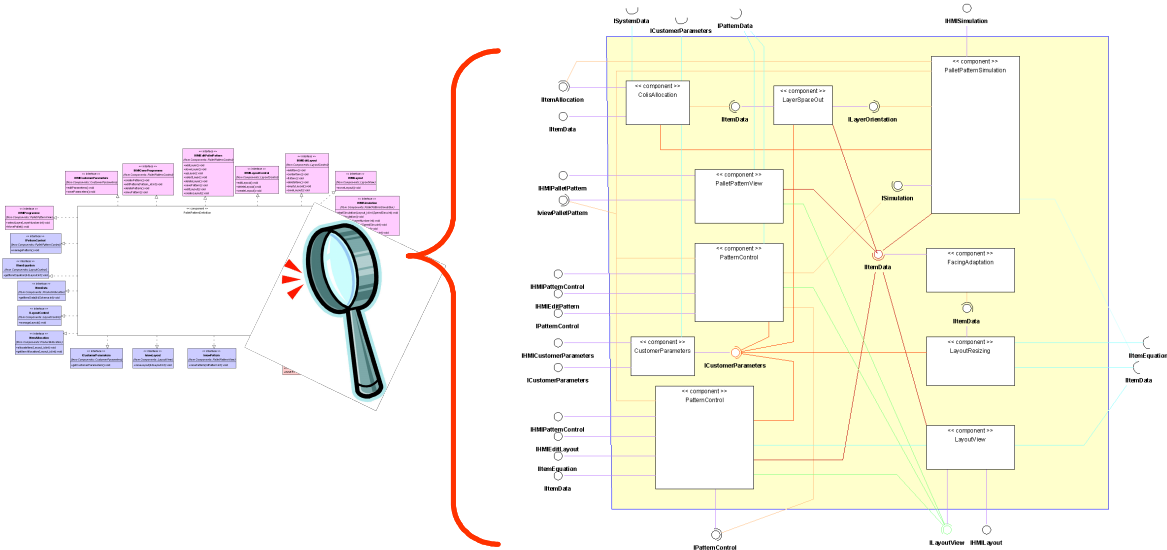


Figure 5: vue WhiteBox

Règles de modélisation:

1. pour accroître la lisibilité on pourra utiliser des couleurs différentes pour les liaisons interfaces fournies et requises
2. de façon systématique les interfaces de type IHM sont traversantes : elles sont visibles à l'extérieur du composant englobant (sur sa vue ne boîte noire) quel que soit la profondeur à laquelle cette interface est réalisée
3. on s'assurera que l'ensemble des interfaces fournies et requises présentes sur la BlackBoxView se retrouvent bien sur la WhiteBoxView et sont reliées par délégation à l'un quelconque des composants internes

3.3 Représentation de la plate-forme cible

Les sections précédentes aboutissent à la construction d'un modèle indépendant de la plate-forme cible (cf. la discussion sur le MDA au début de ce papier). Un tel modèle abstrait doit ensuite être projeté dans le contexte d'une plate-forme d'exécution ciblée. Cette section s'intéresse à la modélisation de la couche matérielle du système CLIPS. Cette modélisation s'articule autour de deux éléments :

- une taxinomie des éléments physiques potentiellement utilisables dans CLIPS – robots, capteurs, etc.
- l'introduction de composants hybrides assurant l'interface entre logiciel et matériel. Ces composants appelés IPP offrent une abstraction intéressante pour encapsuler à la fois une partie du processus métier et des schémas de communication entre logiciel et matériel.

Taxonomie des éléments physiques

Les éléments physiques susceptibles d'être utilisés pour la constitution de la chaîne de palettisation de CLIPS sont modélisés d'une façon homogène, en établissant une taxinomie des matériels.

Le diagramme de classe PhysicalTaxonomy représente une hiérarchie de classes représentant les différents types d'éléments physiques que l'on peut rencontrer, chacun définissant sous forme d'interfaces les services attendus de ces éléments physiques. Les équipements utilisés dans la construction du système physique devront se conformer aux spécifications fournies par cette taxonomie. Ceci permet d'introduire un niveau supplémentaire d'indépendance avec le matériel et améliorer la généricité et la flexibilité du système de supervision.

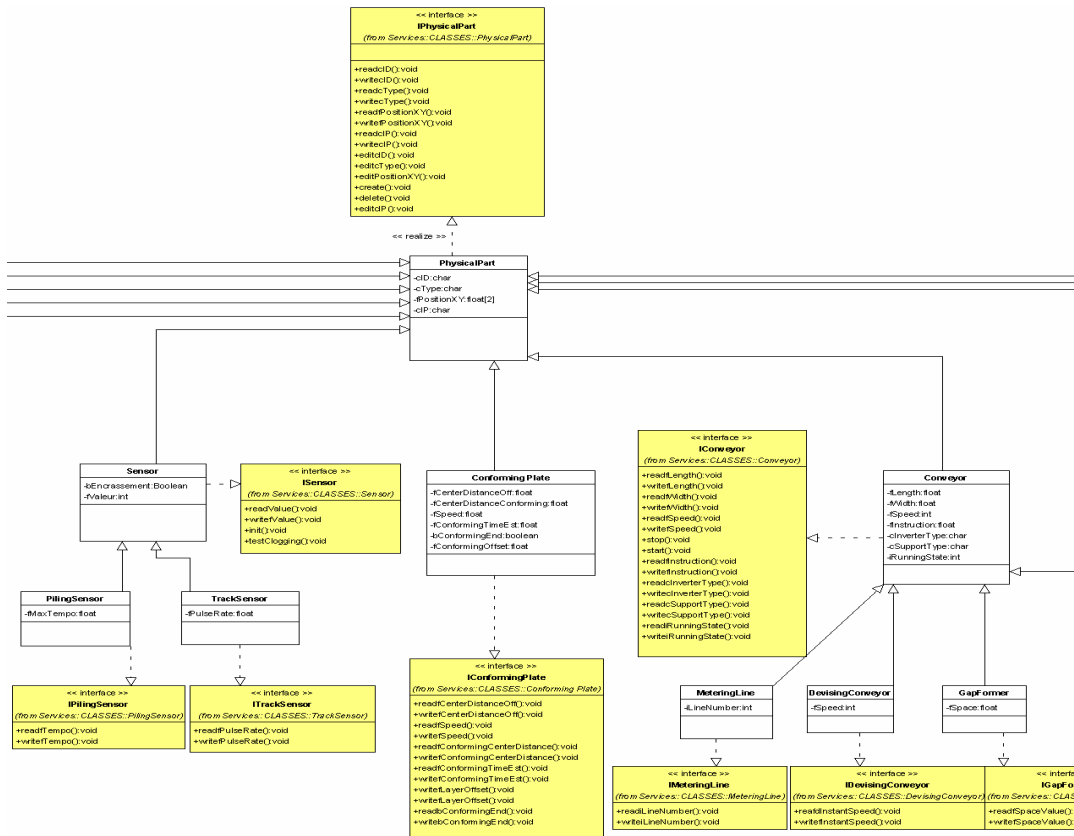


Figure 6: taxonomie des éléments physiques (vue partielle)

3.4 Modélisation du comportement et projection sur la plate-forme cible

Cas général

De façon générale les composants dans CLIPS ont un comportement modélisé à l'aide :

- d'un ensemble de diagramme de séquence expliquant l'enchaînement des appels entre services fournis et requis à l'interface du composant
- d'une machine à état UML2, représentant une succession d'états atteignables par des transitions étiquetées par des gardes et des actions (appels de service, envoi de messages, génération d'événement), prises dans les opérations possibles fournies par le composant sur ses interfaces ou parmi des opérations internes non visibles de l'extérieur du composant.

L'algorithme des opérations est représenté sous forme d'un diagramme d'activité dans lequel les nœuds sont des instructions décrites dans le langage de programmation choisi, les transitions pouvant porter des conditions ou des branchements.

La projection vers la plate-forme logicielle se fait en traduisant ces modèles sous forme de code compilable dans la plate-forme OpenComponents®¹.

Ceci repose sur une correspondance claire entre machine à état et modèle d'exécution d'un composant OpenComponents®. Du code « glue » devra être ajoutée pour se conformer à ce modèle d'exécution. Le développement du code et l'étude de la génération automatique de squelettes de code à partir des modèles UML préciseront cette correspondance dans les prochaines étapes du projet.

Cas particulier des composants d'interface hybrides (composants IPP)

Ces composants ont une modélisation particulière. Leur implémentation dans le système CLIPS suppose une double projection (cf. figure 7):

- en tant que composants logiciel dans la plate-forme OpenComponents® pour assurer des fonctions de suivi du processus métier associé (et remontée des informations lues par les capteurs au réseau de suivi de campagne)
- en tant que bloc fonctionnel automate implanté dans un PLC réalisant le pilotage explicite des éléments physiques impliqués dans le processus métier associé.

Exemple : l'IEP Cadenceur se retrouve sous forme : 1) d'un bloc fonctionnel automate implanté dans un PLC cible et dialoguant avec le convoyeur et les capteurs impliqués dans le cadencage des colis pour le palettiseur ; 2) d'un composant OpenComponents® qui, via un serveur donné (cf. mécanismes de communication section suivante), lit, modifie les paramètres du bloc fonctionnel associé et le cas échéant relaie des messages d'alerte lorsque le fonctionnement sort du cadre nominal.

Pour réaliser cela, les machines à état doivent être enrichies des informations portant sur le fonctionnement nominal des éléments physiques associés au bloc fonctionnel (plage de valeur des capteurs par exemple), de façon à ce que la machine à état comporte des états d'erreur dont la surveillance des conditions de transition permette de relayer au reste du système l'occurrence d'une anomalie.

Cette machine à état sera :

- d'une part partiellement utilisée pour définir le Grafset ou Ladder du bloc fonctionnel automate associé (en l'expurgeant des informations de fonctionnement nominal)

¹ OpenComponents® est une plate-forme de composants développée par Sinovia dédiée à différents types d'applications logicielles. Elle vise une large gamme d'applications industrielles dont la supervision et le contrôle de processus industriels.

- d'autre part projeté en boucle de code pour le composant OpenComponents® lui permettant de suivre l'évolution des états du bloc fonctionnel associé ; ici les états d'erreur prennent toute leur importance.

La génération automatique de squelette de code et de schémas de description des blocs fonctionnels automates sera étudiée parallèlement au développement manuel du code à partir des modèles.

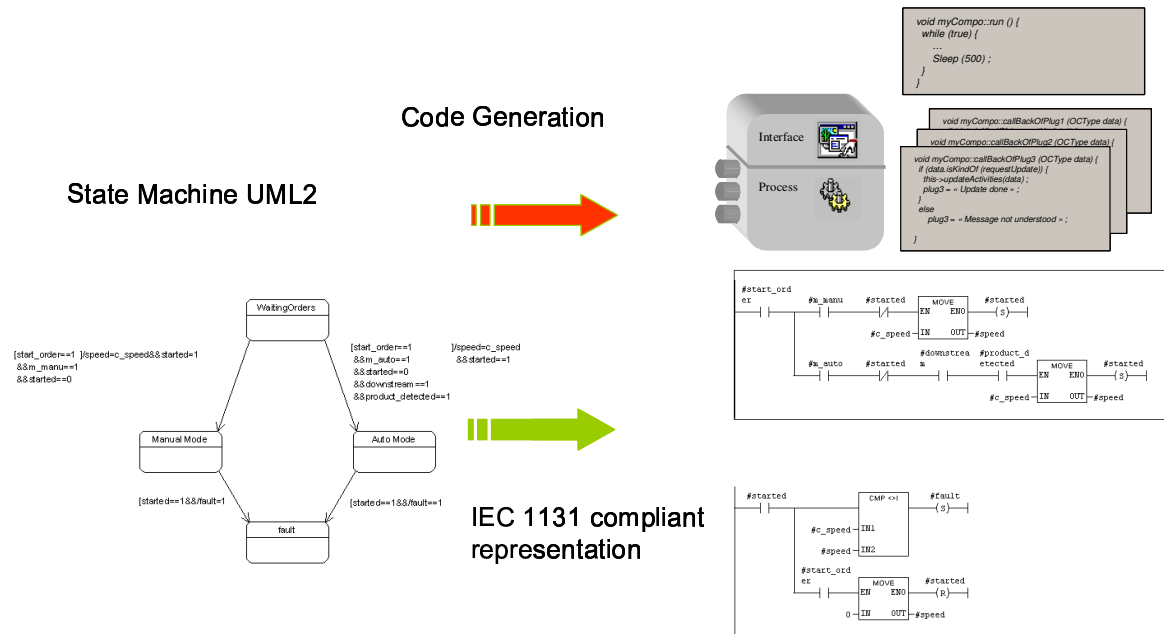


Figure 7: aperçu de la chaîne de génération de code

4. Conclusion

Les spécifications du projet applicatif de palettisation du projet CLIPS ont été réalisées en utilisant la méthodologie partiellement décrite ici. Cette démarche a permis de clarifier la définition des différents sous parties du système et leurs interactions. Cette approche dirigée par les modèles permet d'obtenir un degré de généricité bien plus important que les approches couramment utilisées jusque là dans le domaine des systèmes de production. Des travaux en cours visent à construire une chaîne d'outil supportant cette méthodologie et fournissant des étapes de génération de code automate et logiciel ainsi qu'une assistance à la conception.

Nous pensons qu'une telle approche peut être appliquée à d'autres domaines d'application pour l'automatisation des systèmes de production.

5. Bibliographie

- [1] Arthur J., Roch Jr., « Flexible machining in an integrated system ». In Design and analysis of integrated manufacturing systems, National Academy Press, New York, 34-45. (1988)
- [2] Alan W. Brown, Kurt C. Wallnau, « Engineering of Component-Based Systems ». Computer Society Press, Los Alamitos, CA. (1996)
- [3] OMG, « OMG Unified Modeling Language Specification », version 1.5, (2003)
- [4] OMG, «UML 2 Superstructure Final Adopted Specification », (2004)
- [5] OMG, « UML 2 OCL Final Adopted Specification », (2004)

- [6] OMG, *MDA Guide Version 1.0.1.*, OMG. (2003)
- [7] Sébastien Gérard, et al. Efficient System Modeling of Complex Real-time Industrial Networks Using The ACCORD/UML Methodology, in *Architecture and Design of Distributed Embedded Systems (DIPES 2000)*, Paderborn University, Germany: Kluwer Academic Publishers (2000)
- [8] S. Gérard, F. Terrier, and Y. Tanguy, Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML. In *OOIS'02-MDSD*. 2002. Montpellier: Springer (2002)
- [9] C. Mraidha, S. Robert, S. Gérard, and D.Servat, MDA Platform for Complex Embedded Systems Development, in *Proceedings of Design Methods and Applications of Distributed Embedded Systems*, Toulouse, France (2004)
- [10] Garlan, D., and Shaw, Mary. *An Introduction to Software Architecture*. (CMU/SEI-93-TR-33). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, December 1993. Also in Ambriola, V.; and Tortora, G. (eds.), *Advances in Software Engineering and Knowledge Engineering*, Volume I. Singapore: World Scientific Publishing, 1993.
- [11] Clements P, A Survey of Architecture Description Languages, *Proceedings of the 8th International Workshop on Software Specification and Design*, pp. 16, IEEE Computer Society Washington, DC, USA.
- [12] OMG, *SysML Specifications v0.90 (Draft)*, OMG, 2005.
- [13] Servat D., « Guide méthodologique de modélisation par composants pour la plateforme Plug&Net Open Components », Rapport DTISI/SOL/05-70, Projet RNTL CLIPS
- [14] SINOVA, « OpenComponents® White Paper », Technical Report, available at <http://www.sinovia.com> (in French).