Formalisation des modules perceptifs pour les systèmes de vision réactifs

Soraya Arias, Ève Coste-Manière, Gérard Giraudon, Thierry Viéville

INRIA Sophia
2004 Route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France
e-mail: Firstname.Name@sophia.inria.fr

Résumé Cet article met l'accent sur la nécessité d'introduire un contrôle structuré de l'algorithmique mise en jeu dans les systèmes de vision. Pour spécifier, valider et implémenter ces systèmes, nous proposons de décomposer les actions mises en oeuvre selon deux niveaux d'abstraction : les *actions simples*, qui constituent les briques de base de notre méthodologie, sont caractérisées par un traitement sur une structure de données et par le comportement logique associé qui en rythme l'exécution ; au niveau supérieur, le contrôleur global du système est obtenu en combinant ces actions simples en un ensemble ordonné traduisant la logique du système selon un séquencement de "modes de marche". Dans les deux cas, on considère la logique du système dans le cadre formel des systèmes réactifs et du langage impératif synchrone ESTEREL. La réalisation d'une application de détection et suivi de cible valide expérimentalement cette méthode.

Mots-Clés : architecture logicielle, système de vision, réactivité, Esterel, validation

1 Introduction

Considérons une application de vision dont le but est de suivre un objet en mouvement. Pour réaliser ce suivi, une phase de détection de l'objet (lissage, détermination des régions en mouvement, sélection de région) et une phase de déplacement de la caméra (calcul du déplacement de la caméra, commande de la caméra) sont nécessaires. De manière classique, tous les modules de calcul utilisés sont définis, un à un, "à la main" puis combinés afin de déplacer la caméra, d'acquérir une image, de la lisser, d'y rechercher les zones en mouvement, et de sélectionner la cible puis de la suivre. L'exécution de chacune de ces étapes, et par conséquent de l'application, dépend de nombreuses conditions : certaines doivent être satisfaites avant le démarrage des calculs (e.g. caméra en marche, ordre utilisateur...), d'autres peuvent signaler des dysfonctionnements (e.g. luminosité défaillante...), enfin certaines indiquent que les objectifs ont été atteints (e.g. objet en mouvement détecté...). Ces différentes conditions caractérisent, à faible bande passante, les "instants discrets" qui rythment l'évolution des "flots de données continus", à grande bande passante, transformés par les différents algorithmes de calcul mis en oeuvre par le système de vision. Ce qui découple ainsi les aspects réactifs des aspects algorithmiques de ce système.

Par conséquent, pour spécifier, valider et implémenter les systèmes de vision, il est important de traiter en les combinant, ces aspects continus et discrets, à l'aide d'outils théoriques dédiés. Une brève analyse des approches traditionnelles utilisées en vision (cf section 2) révèle que bien souvent ces deux aspects ne sont pas appréhender dans un cadre formel, limitant ainsi les possibilités d'analyse formelle et de validation des systèmes mis en oeuvre. Pour compenser, nous proposons une méthodologie de programmation qui gère, en harmonie avec les aspects algorithmiques, les spécificités inhérentes au *contrôle* des systèmes de vision (cf section 3) dans le cadre formel des systèmes réactifs et du langage impératif synchrone ESTEREL (cf section 4). La réalisation pratique d'une application de suivi de cibles illustre la globalité de notre approche (cf section 5).

2 État de l'art

L'approche traditionnelle introduite par Marr [1] considère la vision comme "passive", c'est-à-dire comme une suite de transformations fondées sur un modèle mathématique, permettant de passer d'une représentation à une autre.

En opposition, Aloimonos propose dans [2] la notion d'*observateur actif* et la notion de mouvement contrôlé pour contraindre les hypothèses restrictives de la vision passive.

De même, Bajscy [3] pose le problème de la vision comme un problème de contrôle et de *perception active*: à la définition algorithmique des modules de traitement, elle rajoute un processus de décision et de contrôle. Elle va jusqu'à donner des consignes pour définir de manière explicite, en relation avec la partie contrôle, les paramètres et les erreurs qui interviennent dans les traitements d'image et sur les capteurs.

D'un autre côté, des efforts de méthodologie de programmation existent dans la communauté vision avec : IUE (Image Understanding Environnement)[4], ou Target Jr [5]. Ces environnements ne proposent pour l'instant que la définition des structures des données et d'algorithmes standards pour les capteurs et le traitement d'image. La notion de contrôle y est diffuse.

L'idée est alors de proposer une méthodologie pour faciliter la programmation des systèmes de vision permettant d'associer un contrôle aux algorithmes de traitement utilisés.

3 Position du problème

L'expérience actuelle acquise à travers différentes applications (l'interprétation en imagerie aérienne [6], [7], la robotique mobile [8], [9] ou la reconnaissance d'objets sur séquence d'images [10] ou [11]), permet de faire une analyse générique plus pertinente des contraintes qu'un système de vision doit respecter pour être valide. Ce sont entre autres :

- la prise en compte du capteur visuel et de son contrôle dans le processus de vision [3], [2] ;
- le contrôle et paramétrage des algorithmes pour, par exemple, choisir l'opérateur de traitement le mieux adapté à la résolution d'un problème donné [8] ou encore adapter le paramétrage en fonction de l'état du système et de l'utilisateur [12] ;
- le contrôle du flot de données dans le but de sélectionner les données les plus pertinentes parmi celles disponibles [8], [9], [13] ;
- le respect des contraintes temporelles dans les environnements dynamiques [10], [11].

Pour répondre de manière à satisfaire ces critères et à assurer un contrôle robuste et fiable de l'exécution des algorithmes qui composent un système de vision :

- la **spécification** des entités de vision doit tenir compte à la fois du traitement algorithmique des données manipulées et du contrôle de ces algorithmes. Ces deux aspects doivent être intimement couplés et ceci de manière transparente pour l'utilisateur final.
  Le processus de spécification doit être incrémental, pour permettre l'élaboration pas à pas du système et hiérarchique, pour maîtriser sa complexité ;
- la **validation** du système doit assurer la conformité de sa programmation avec ses contraintes. Il faut donc utiliser, autant que faire se peut, des modèles mathématiques qui permettent d'analyser et valider formellement la spécification du système et par conséquent assurer un fonctionnement fiable ;
- l'**implémentation** doit mettre en œuvre les spécifications de manière efficace en assurant entre autre le respect des contraintes temporelles et la gestion du caractère concurrent et distribué d'un système de vision. Le passage de la spécification à l'implémentation doit s'effectuer de manière transparente pour l'utilisateur, en utilisant, le plus possible, la génération automatique du code.

En conséquence, l'objectif est de proposer une méthodologie formelle complète intégrant une partie spécification, validation et implémentation des entités de vision nécessaires à la réalisation d'applications complexes, comme illustré au §5.

4 Méthodologie proposée

4.1 Principe Général

La méthodologie proposée part du constat que, pour la spécification d'un système de vision pour lequel on souhaite traiter, dans un formalisme cohérent, à la fois les problèmes algorithmiques et de contrôle, il existe un niveau où la logique du système peut être décrite comme un séquencement de "modes de marche" : chaque mode encapsulant des algorithmes de traitement sur le flot de données nécessaires à son déroulement.
Si on reprend l'exemple proposé en introduction, on pourra considérer une phase de détection d'objets statique, et une phase de détection d'objets avec suivi. Pour chaque phase, il va falloir décider des *préconditions* suivant lesquelles un traitement peut être lancé (pour la détection de cible, la caméra doit être en état de marche), des actions à lancer si une *exception* se produit au cours d'un traitement (la caméra tombe en panne, alors on arrête le traitement en cours et on lance un traitement d'urgence), des *postconditions* qui indiquent la fin d'un traitement (on ne lancera la phase de déplacement de la caméra que si la cible sort du champ visuel). Mais on peut aussi décider comment on va agencer les phases de détection statique et celle avec suivi : on peut lancer la phase de suivi de cible et décider de passer à la phase de détection statique si l'exception `le mouvement de la caméra est défectueux` survient. En fait, l'ensemble, précondition, exception, postcondition, lorsqu'il est bien analysé, définit un comportement logique associé à l'algorithmique du système.

Deux entités principales sont alors proposées afin de spécifier de manière hiérarchique et structurée un système de vision :

- l'*action de vision élémentaire* qui constitue la brique de base de notre méthodologie.
  Elle se caractérise par un traitement sur une structure de données et par un comportement logique associé à ce traitement. Ce comportement est rythmé par les événements qui indiquent la satisfaction des préconditions de démarrage, des postconditions ou des exceptions du traitement algorithmique associé.
  Par exemple : pour l'action "détection de cible en mouvement", le comportement logique pourra être défini avec : la précondition de démarrage, `cible à détecter choisie` ; pendant le déroulement de l'action, les exceptions suivantes à gérer `image acquise vide`, `calcul n'atteint pas le seuil prédéfini` ; la postcondition de fin de l'action, `la cible n'a pas bougé depuis x instants`.
- l'*ensemble ordonné des actions élémentaires* qui se caractérise par l'enchaînement logique de ces actions élémentaires afin d'aboutir à la spécification complète d'un système de vision.
  Par exemple : si on souhaite faire une application de détection de cible, on peut séquencer les actions "calibration de la caméra" et "détection de cible". Au niveau logique de cette application, on va définir la précondition `caméra en marche`, la postcondition `arrêt demandé par l'utilisateur`, et l'exception `problème de calibration de la caméra` : si elle survient pendant la phase de calibration, on arrête l'application, si elle survient pendant la détection, on stoppe cette action et on relance l'action de calibration suivie de l'action de détection.

On considère la "logique" du système dans le cadre formel des systèmes *réactifs* [14]. Le comportement est alors entièrement spécifié de manière déterministe par l'ensemble des séquences permises d'événements externes d'entrée et sortie qui constitue l'interface du système réactif avec son environnement.

Par cette définition formelle, on se différencie de la notion de comportement lié au domaine de l'intelligence artificielle. Dans ce dernier cas, ce sont les "connaissances" que l'on peut avoir sur le système de vision qui vont servir de base pour faire de la supervision, et de la planification de programmes à l'aide de systèmes experts [15], ce qui ne permet pas d'analyse formelle.

En revanche, le type de modélisation du comportement que nous proposons nous permet d'assurer une spécification de la logique du systèmes non-ambigüe et permet des vérifications formelles avant l'exécution.

Dans le cadre de la programmation des *systèmes réactifs*, nous nous intéressons à la famille des langages synchrones [16] qui pourraient être regroupés en deux catégories : les "impératifs" et les "flots de données" [17]. L'objectif, dans ce travail, est de spécifier les aspects logiques et comportementaux des systèmes de vision, par conséquent les langages de type impératif sont mieux adaptés pour la spécification des aspects "discrets" que les langages flot de données mieux adaptés pour la spécification des aspects "continus" [18].

Parmi les langages synchrones impératifs, nous avons choisi d'utiliser le langage ESTEREL. Il est basé sur une sémantique mathématique forte et la compilation d'un programme produit un automate fini déterministe qui peut être ensuite analysé par des outils de vérification dédiés aux automates tel que FC2TOOLS [19]. L'utilisation de ce langage pour programmer les applications de vision rend possible la vérification des propriétés logiques associées. Si on reprend l'exemple cité plus haut, on pourra vérifier que pour toute exécution du programme, l'action "détection de cible" va être désactivée chaque fois qu'un problème de calibration est détecté.

Cependant, il est nécessaire de pouvoir contraindre la programmation ESTEREL pour automatiser l'élaboration et faciliter la validation des applications. L'idée est alors de s'inspirer d'une méthodologie solide conçue dans le domaine des applications robotiques (la méthodologie ORCCAD [20], le langage de programmation MAESTRO [21]) en intégrant les spécificités liées au domaine de la vision citées au §3.

4.2 Les outils : ORCCAD, MAESTRO, ESTEREL

ORCCAD (**O**pen **R**obot **C**ontroller **C**omputer **A**ided **D**esign) [22], est un environnement conçu pour spécifier, valider avec des méthodes formelles, simuler et implémenter les applications robotiques.

Du point de vue conceptuel, ce système considère principalement les deux niveaux d'abstraction suivants :

1. le niveau fonctionnel.
   On considère ici le formalisme de la **tâche robot** qui est définie comme la spécification complète et paramétrée :
   - d'un ensemble d'algorithmes qui servent à décrire la loi de commande associée invariante pendant sa durée de l'exécution.
   - d'un comportement logique déterminé par un ensemble de signaux qui peuvent survenir avant, pendant et après l'exécution de la TÂCHE ROBOT. Ces signaux sont typés (préconditions, exceptions et postconditions).

   A chaque TÂCHE ROBOT, on associe une *vue abstraite* qui synthétise les caractérisques de son comportement logique, la ressource physique utilisé (e.g. le nom du robot) et ses paramètres.

   La partie implémentation des TÂCHES ROBOT est réalisée à travers les **tâches modules**. Ces TÂCHES MODULES sont des tâches temps réel qui implémentent de manière optimale les fonctions élémentaires qui interviennent dans les TÂCHES ROBOT.

2. le niveau contrôle.
   Ce niveau considère des **procédures robot**.
   Ces PROCÉDURES ROBOT sont la représentation de la spécification d'un arrangement au niveau logique et temporel de plusieurs TÂCHES ROBOT, dans le but de réaliser un objectif global, ou plus généralement une application.
   On va utiliser le langage de programmation de mission MAESTRO [23] pour définir ces PROCÉDURES ROBOT. Ce langage utilise directement les vues abstraites des différentes TÂCHES ROBOT et permet grâce à une syntaxe définie dans [21] de les arranger pour spécifier différentes PROCÉDURES ROBOT et élaborer finalement une mission robotique. Une fois la mission programmée, le programme MAESTRO est transformé en un automate de contrôle.
   Cet automate pourra être ensuite utilisé pour établir des vérifications du comportement de l'application et pour prouver que certaines propriétés des spécifications sont bien vérifiées.

4.3 Adaptation à la vision

La méthodologie que nous envisageons pour les systèmes de vision est basée sur celle de ORCCAD. Par conséquent, on établira une analogie TÂCHES ROBOT → TÂCHE VISION et PROCÉDURES ROBOT → PROCÉDURES VISION. En effet, on peut considérer que pour la TÂCHES VISION, la "loi de commande" correspond à la transformation mathématique faite sur le flot de données continus (image, liste chaînée des contours, ...) et que le comportement correspond au flot de données discret qui donne des informations sur la validité, la cohérence du flot de données continu.

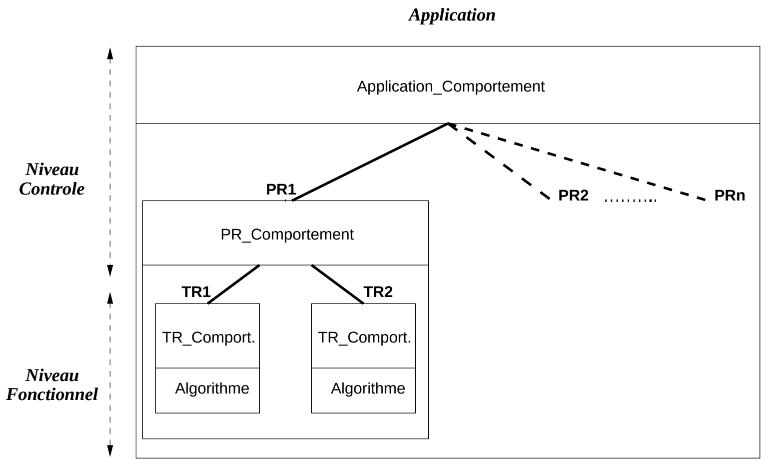

FIG. 1 – *Hiérarchie des Spécifications dans* ORCCAD

Par rapport aux contraintes citées au §3 :

- au niveau de la gestion des paramètres :
  on a la possibilité de paramétrer les TÂCHES VISION. Chaque TÂCHE VISION aura une liste de paramètres associée qui pourront être modifiés soit à partir d'une interface utilisateur soit par une autre TÂCHE VISION soit encore par l'intermédiaire d'un signal particulier provenant de l'automate qui entraînera le reparamétrage de la TÂCHE VISION. Selon les besoins de l'utilisateur, il pourra adopter l'une ou plusieurs de ses possibilités pour le paramétrage.
- au niveau du capteur de vision :
  ORCCAD permet d'associer à chaque TÂCHE VISION une ressource physique qui correspond à la spécification du capteur envisagé. Cette ressource sera contrôlée à travers le comportement logique de la TÂCHE VISION associée.
- au niveau des contraintes de temps :
  ORCCAD permet de définir la fréquence d'appel de la partie algorithmique d'une TÂCHE VISION. Ainsi, on peut spécifier la cadence d'exécution de l'application.

Cependant, il serait intéressant de confondre l'intérêt de la spécification des contrôleurs multicadence désynchronisés produits par ORCCAD (nécessaires dans le cadre robotique) aux besoins moins restrictifs des applications de vision et étudier l'utilisation des langages synchrones flots de données comme LUSTRE ou SIGNAL. Ceci permettrait de spécifier et implémenter dans le même formalisme, à la fois les aspects flot de données et contrôle. Dans ce cadre, les propriétés à vérifier pourraient exprimer des relations entre les aspects logiques de l'application et les aspects calculatoires des algorithmes.

5 Réalisation d'un prototype

Nous illustrons notre approche par la réalisation pratique d'une application de suivi "temps-réel" de mouvements.
Cette application a pour but de suivre un objet prédéterminé par un utilisateur tout en effectuant la détection d'autres objets entrant dans la scène observée. Cette dernière action permettra de faire basculer le suivi d'un objet sur un autre : soit par intervention directe de l'utilisateur, soit lors de la non-réalisation d'un critère de mouvement de l'objet suivi.

5.1 Spécification des tâches de vision

Comme explicité au paragraphe 4.1, l'action de vision élémentaire se compose d'une partie algorithmique et d'une partie comportement logique associé.

Nous allons préciser pour chacune de ces parties les principales caractéristiques mises en oeuvre dans l'application.

La plateforme expérimentale Nous avons utilisé la plateforme ARGÈS [24]. Elle comporte les éléments suivants : (1) une caméra couleur contrôlée par ordinateur ; (2) une tourelle Pan-Tilt avec une interface RS232C, pour piloter le moteur ; (3) un rail de $85cm$ sur lequel peut se déplacer la tourelle Pan-Tilt ; (4) un panneau d'acquisition et de compression SunVideo basé sur la librairie XIL. Le système est piloté à travers un driver logiciel accessible à travers un fichier interface, pour permettre de commander la caméra, modifier le pan, le tilt, la position de la tourelle sur le rail.

La partie algorithmique de vision Nous utilisons les méthodes décrites dans [25] pour la détection et le suivi de cibles. Par conséquent, la base algorithmique utilisée est la même qu'expliquée dans [25], à savoir :

- détection et suivi d'une cible sans mouvement de la caméra : acquisition d'image, lissage, différence de l'image à t-1 avec celle à t, seuillage, segmentation de zones de mouvement, sélection de l'objet cible.
- détection et suivi d'une cible avec mouvement de la caméra : même phase que précédemment en rajoutant la stabilisation après la phase de lissage et le suivi et le déplacement de la caméra s'il y a lieu après la phase de sélection de cible.
- la calibration automatique de la caméra sur objet fixe.

La figure 2 montre comment définir une TÂCHE VISION sous ORCCAD.
Chaque boîte représente une fonctionnalité telle que le lissage (boîte notée `Smooth`), la différence d'image (boîte notée `Difference`). Il existe 2 boîtes plus particulières : la boîte de la ressource physique, notée `VideoCamera` pour gérer le capteur caméra, et la boîte de l'automate, notée `AUTOTRACK` qui représente l'automate de contrôle des événements propres à la TÂCHE VISION
On relie les boîtes les unes aux autres par des ports. Il en existe de 4 types : les ports de données symbolisés par des ronds, les ports des ressources physiques symbolisés par des rectangles, les ports de paramètres symbolisés par des étoiles et les ports d'événements symbolisés par des losanges.

Sous l'environnement ORCCAD, chaque MODULE de vision requiert la définition de fichiers nécessaires à son implémentation, comme `var.c` pour la déclaration de variables, `init.c` pour initialiser la après la réception des préconditions, `compute.c` pour le corps principal de TÂCHE VISION, et `end.c` pour le traitement à faire une fois les postconditions reçues.
La figure 3 illustre la spécification sous ORCCAD du MODULE lissage.

La partie comportement logique L'interface avec le niveau hiérarchique supérieur se fait à travers la définition des vues abstraites des actions élémentaires. Ces vues abstraites sont utilisées directement par les PROCÉDURES ROBOT, et par le langage MAESTRO. La vue abstraite contient les informations qui définissent le comportement de l'action qu'elle représente, comme les préconditions, les postconditions, les exceptions.
La figure 4 illustre la vue abstraite de l'action détection et suivi de cible.

5.2 Spécification de l'application

Notre application gère à travers une interface le lancement de 4 actions différentes : (1) la commande du pan de la caméra ; (2) la commande du tilt de la caméra ; (3) la commande du rail de la caméra ; (4) la détection et suivi de cible. La spécification de cette partie principale correspond à la fenêtre notée `(a)` dans la figure 5.

Plusieurs cas de figure sont gérés :

1. si un problème au niveau des paramètres de la caméra vient à se produire, la phase de détection et de suivi de cible est stoppée et est replacée par le séquencement d'une phase de recalibration de la caméra puis de la détection et suivi de cible ;

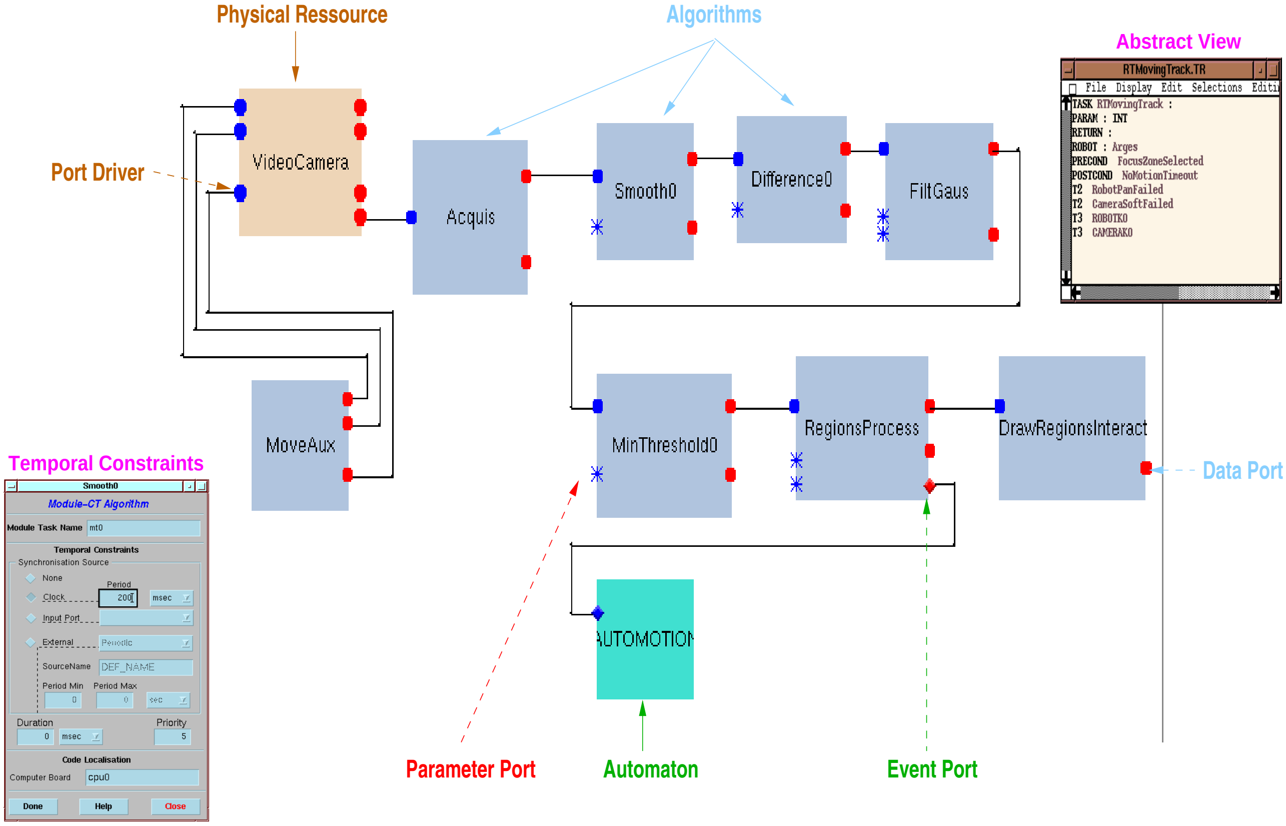

FIG. 2 – *Spécification d'une tâche vision sous* ORCCAD

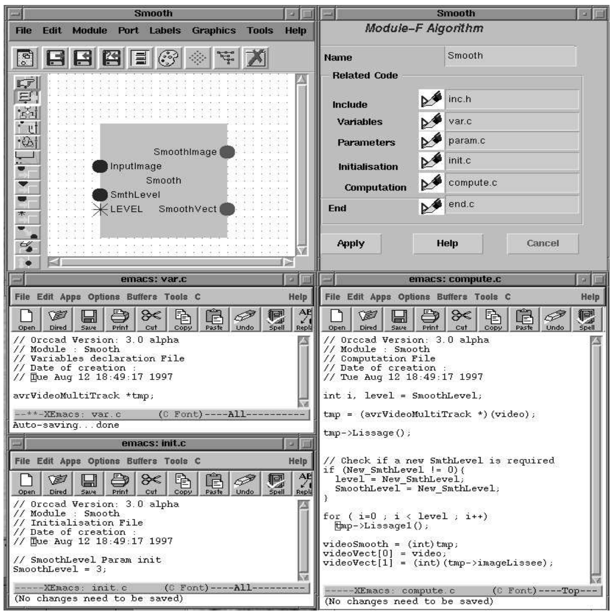

FIG. 3 – *Spécification du* MODULE *lissage avec les outils* ORCCAD

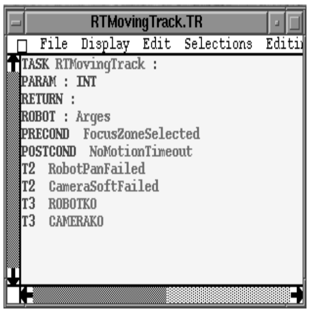

Fig. 4 – *Vue abstraite de l'action détection et suivi de cible*

2. si un nouveau problème au niveau de la paramétrisation de la caméra vient à se reproduire, seules les commandes relatives à la caméra sont disponibles à l'utilisateur (voir la fenêtre notée `(b)` dans la figure 5) ;
3. si un problème au niveau de la robotique de la caméra survient, la phase de détection et de suivi de cible est remplacée par une phase de détection statique de cible (voir la fenêtre notée `(c)` dans la figure 5) ;
4. si le robot-caméra ou la caméra tombe en panne, l'application se termine immédiatement. (voir le bas de la fenêtre notée `(a)` dans la figure 5).

Ce scénario a été écrit par l'intermédiaire d'un programme Maestro présenté ci-dessous :

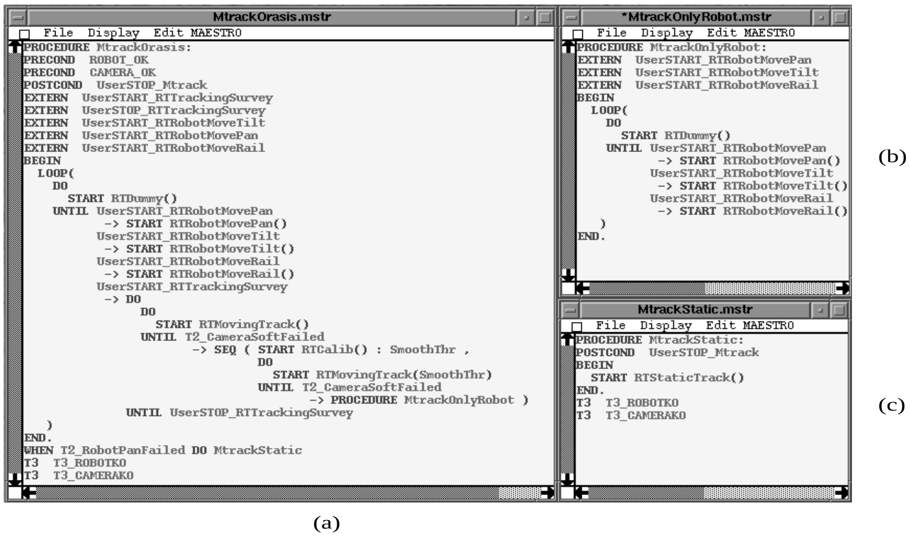

Fig. 5 – *Spécification de l'application avec Maestro*

5.3 Vérification

Dans le paragraphe précédent, nous avons présenté comment on spécifie à l'aide du langage Maestro l'application. Le programme ainsi défini est transformé en Esterel, et en analysant l'automate produit lors de sa compilation, on vérifiera et validera un certain nombre de propriétés logiques sur l'application.

Ce travail se fait avec les outils de vérification automatique dédiés à l'analyse d'automates finis : fc2tools, ATG. On peut donc grâce à ces outils détecter des comportements anormaux, ou vérifier qu'une propriété particulière est bien réalisée.

La figure 6 illustre une partie de l'automate, élaboré à travers le langage de programmation de mission et visualisé grâce à l'outil ATG. Il correspond à la partie de contrôle de la requête de détection et de suivi de mouvement. Au niveau de la lecture de l'automate, les labels préfixés par ? correspondent aux signaux d'entrée, et ceux préfixés par ! correspondent aux signaux de sortie.

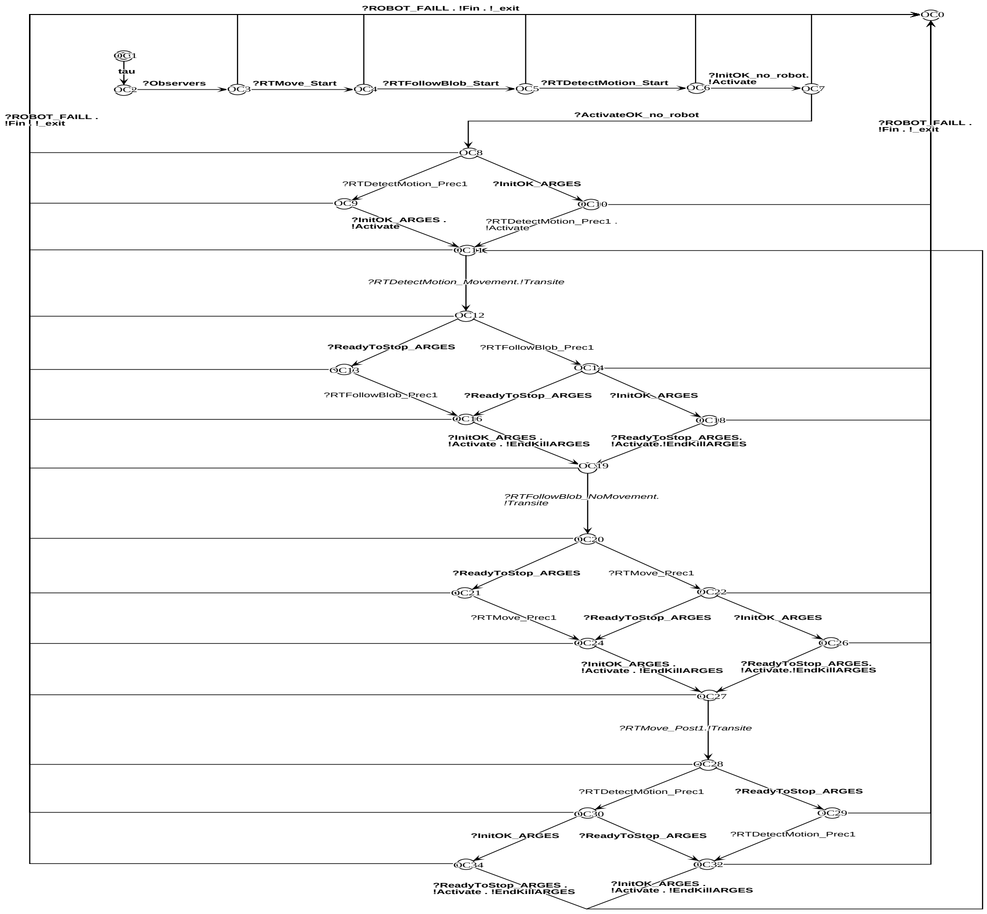

FIG. 6 – *Fragment d'automate généré par* ESTEREL

5.4 Implémentation

Un fois, l'application globale spécifiée, le code exécutable du prototype est généré automatiquement par l'environnement ORCCAD. Ce code gère entièrement la synchronisation du séquencement des différentes TÂCHES VISION spécificiées par l'utilisateur et la stratégie définie en MAESTRO.

Le système d'exploitation sur lequel tourne l'application est un environnement SOLARIS. Pour cela, nous avons porté la partie qui gère le noyau temps-réel et le générateur de code du système ORCCAD sous SOLARIS (utilisation des sémaphores et de threads Solaris, queues de messages IPC, gestion de timers Posix).

5.5 Résultats

Apport de cette méthode L'utilisateur dispose d'une interface avec laquelle il ne va définir de manière incrémentale que les modules qui composent l'application qu'il a décidé de réaliser.
Cette méthode propose différents degrés d'utilisation suivant le niveau de compétence de l'utilisateur : si c'est un spécialiste en algorithme de vision, il va pouvoir concevoir sa nouvelle fonctionnalité sous la forme d'un nouveau MODULE ; si c'est un utilisateur qui cherche à créer une application de vision, il va se servir

de la batterie de MODULES déjà définie et élaborer les TÂCHES VISION qui composent son application. Ceci permet une réutilisabilité plus facile des MODULES de vision.
La gestion du contrôle des actions et de l'application se fait de manière simple et assez intuitive : soit à travers la définition, pour chaque action, des événements et des paramètres que l'utilisateur décide de gérer pour contrôler son action, soit à travers l'utilisation du langage MAESTRO dont la syntaxe est simple.
L'environnement ORCCAD permet de plus à l'utilisateur de voir de manière transparente l'implémentation de la gestion des contraintes temps-réel. Le code est automatiquement généré pour un système d'exploitation cible, à savoir ici SOLARIS.

Performance temps-réel Du point de vue des performances temps-réel, on obtient des résultats dans le même ordre de grandeur que ceux obtenus par [25], à savoir une fréquence d'acquisition d'images d'environ 10 images par seconde, même si l'utilisation de threads, de sémaphores, de queues de message et de timers produit une surcharge du système. Mais ce qu'il est important de retenir, c'est que cette programmation de l'application, contrairement à celle de [25], gère véritablement les aspects concurrents du système global.

6 Conclusions

On propose une méthodologie qui est incrémentale et dont l'originalité consiste à utiliser un formalisme de contrôle clairement défini et intimement lié avec les aspects algorithmiques du problème, afin de palier les insuffisances des systèmes de vision actuels. Pour ce faire, nous nous fondons sur les concepts introduits par les systèmes réactifs et leur programmation.
Les problèmes inhérents à la formalisation des actions de vision sont analysés et illustrés à travers une application effective de suivi de mouvements. Cette analyse nous permet de proposer et justifier une définition formelle des actions de vision et une méthodologie pour leur programmation, qu'il faudra cependant valider sur d'autres exemples applicatifs.

Références

[1] D. Marr. *VISION*. W. H. Freeman and Company, New York, 1982.

[2] J.Y Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. In *Proceedings of the International Conference on Computer Vision*, pages 35–54, London, June 1987.

[3] R. Bajscy. Active perception. *Proceedings of the IEEE, Special Issue on Computer Vision*, 76(8):996–1005, August 1988.

[4] J. Mundy, T. Binford, T. Boult, A. Hanson, R. Beveridge, R. Haralick, V. Ramesh, C. Kohl, D. Lawton, D. Morgan, K. Price, and T. Strat. The image understanding environment program. In *IEEE Proc CVPR'93*, pages 406–416, New-York, June 1993.

[5] TARGET WEB HOME PAGE. "http://www.esat.kuleuven.ac.be/˜targetjr/".

[6] T. Matsuyama. Knowledge-based aerial image understanding systems and expert systems for image processing. *IEEE Transactions on Geoscience and Remote Sensing*, 25(3):305–316, 1987.

[7] P. Garnesson and G. Giraudon. An image analysis system: Application for aerial imagery interpretation. In *ICPR'90*, Atlantic City, New Jersey, June 1990.

[8] G. Giraudon and F. Sandakly. Interprétation de scènes d'intérieur pour un robot mobile. *Techniques et science informatiques*, 15(6):1–31, 1996.

[9] J. K. Rosenblatt. *A distributed Architecture for Mobile Navigation*. PhD thesis, Robotic Institute, Carnegie Mellon University, January 1997.

[10] S. Estable. *Reconnaissance d'Objets en Environnement Extérieur Dynamique - Application à la Reconnaissance de Panneaux Routiers*. PhD thesis, Université Blaise Pascal - Clermont II, July 1996.

[11] T. Viéville, E. Clergue, R. Enciso, and H. Mathieu. Experimentating with 3D vision on a robotic head. *Robotics and Autonomous Systems*, 14(1), 1995.

[12] W. Schneider, W. Eckstein, and C. Steger. Real-time visualization of the interactive parameters changes in image processing systems. In *Proceedings SPIE conf. 3017, Visual Data Exploration and Analysis IV*, pages 286–295, San Diego, February 1997.

[13] M. Klupsch and W. Eckstein. Object-oriented image processing in SMALLTALK : Using complex operators objects. In *Proceedings International Conference on Information Systems Analysis and Synthesis, ISAS'96, July 22-26*, pages 833–839, Orlando, 1996.

[14] D. Harel and A. Pnuelli. *On the Development of Reactive Systems, Logic and Models of Concurrent Systems*, pages 477–498. Springer-Verlag, 1985.

[15] V. Clément, M. Thonnat, and J. Van den Elst. Supervision of the perception tasks for autonomous systems : the OCAPI approach. Technical Report RR-2000, INRIA, June 1993.

[16] G. Berry. Real-time programming: General purpose or special-purpose languages. In *Information Processing 89*, pages 11–17. Elsevier Science Pblishers B.V. (North Holland), G. Ritter, editor, 1989.

[17] N. Halbwachs. *Synchromous Programming of the Reactive Systems.* Kluwer Academic Publisher, 1st edition, 1993.

[18] E. Marchand, E. Rutten, and F. Chaumette. Applying the synchronous approach for real time active visual reconstruction. Technical Report RR-2383, INRIA, November 1994.

[19] FC2TOOL WEB HOME PAGE. "http://www.inria.fr/meije/verification/".

[20] ORCCAD WEB HOME PAGE. "http://www.inrialpes.fr/iramr/pub/Orccad/".

[21] E. Coste-Manière and N. Turro. The MAESTRO language and its environment : Specification, validation and control of robotic missions. In *Proceedings of the 10th IEEE/RSJ International Conference on Intelligent Robots and Systems*, Grenoble, France, September 1997.

[22] The ORCCAD Team. An integrated and modular approach for the specification, the validation and the implementation of complex robotics missions. *To appear in the International Journal of Robotics Research*, 1998.

[23] N. Turro and E. Coste-Manière. Mission programming language. Technical Report Final Report of the WORKPACKAGE 2 TASK 2.2 of the UNION, Esprit Basic Research Project, July 1996.

[24] ARGES WEB HOME PAGE. "http://www.inria.fr/robotvis/personnel/vthierry/acvis-demo/arges/arges.html".

[25] C. De Murcia, M. Niemaz, and T. Viéville. Détection et suivi de cibles sur une durée indéterminée. In *Proceeding of Orasis'97, October*, La Colle s/Loup, France, 1997.