Rapport de projet De Stijl

version 2 janvier 2017

Mettez ici vos noms et les parties sur lesquelles vous avez travaillées : conception, robot, vidéo, mission, intégration, rédaction du compte-rendu
Exemple : P.-E. Hladik (conception, robot, rédaction du compte-rendu)

— Ce qu'il faut faire —

Remplacez tous les textes en bleu et supprimer les textes en rouge

Le rapport est à rendre en pdf et à envoyer par mail à votre encadrant de TP au plus tard le 20 janvier 2017.

Vous devez aussi rendre votre code (uniquement les fichiers que vous avez écrits ou modifiés) sous la forme d'une archive (zip ou tar).

Vous pouvez utiliser word ou un autre logiciel d'édition pour rédiger ce rapport, par contre vous devez **obligatoirement** respecter la structure décrite ici.

Critères d'évaluation :
- Qualité rédactionnelle,
- Exhaustivité et justesse des règles de codage,
- Qualité de la conception (clarté, respect de la syntaxe, exhaustivité, justesse),
- Qualité des explications,
- Respect des règles dans la production du code

Compétences évaluées :
- rédaction et communication sur un dossier de conception
- concevoir une application concurrente temps réel
- analyser une conception
- passer d'un modèle de conception à une implémentation
- écriture de code C et utilisation de primitives au niveau système

1 Conception

Mettez dans cette partie tous les éléments de votre conception en particulier vos diagrammes AADL (vue globale du système et détails des threads). Cette partie doit être autosuffisante pour comprendre votre application.

Pour faciliter la lecture des schémas, vous allez présenter votre conception en trois parties, l'une focalisée sur la communication entre le moniteur et le superviseur, la seconde consacrée au contrôle du robot et la troisième au traitement vidéo.

La partie de gestion automatique d'une mission du robot sera à part pour ceux l'ayant traitée.

Si vous le souhaitez, au lieu de dessiner vos diagrammes sous un éditeur, vous pouvez joindre un scan de vos schémas — ils doivent être lisibles et propres.

1.1 Diagramme fonctionnel général

Mettez ici un diagramme fonctionnel qui présente les principaux blocs de votre conception. Pour cela, inspirez vous du diagramme ci-dessous (fig. **??**) en indiquant pour chaque groupe de threads les données et ports partagés. La figure **??** a été réalisée à partir du document de conception. **Vous devez absolument conserver le découpage en trois groupes de threads (`th_group_gestion_moniteur`, `th_group_gestion_robot`, `th_group_vision`).**

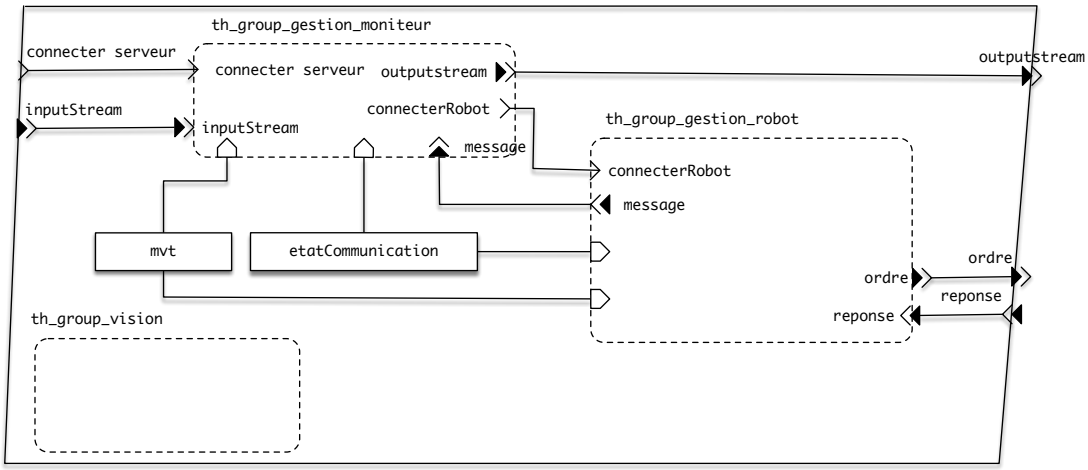

Fig. 1: Diagramme fonctionnel du système

1.2 Groupe de threads gestion du moniteur

Placez ici :

— le diagramme fonctionnel en AADL décrivant le groupe de threads de gestion du moniteur (voir exemple de la figure **??** réalisée à partir du dossier de conception),

— remplir le tableau **??** pour expliquer le rôle de chacun des threads,

— les diagrammes d'activité de chaque thread de ce groupe.

Décrivez tous les éléments (paramètres, variables, etc.) qui vous semblent pertinents pour comprendre les diagrammes.

1.2.1 Diagramme fonctionnel du groupe gestion du moniteur

Exemple de diagramme fonctionnel pour le groupe de thread de gestion du moniteur. Mettez à jour ce diagramme avec votre conception.

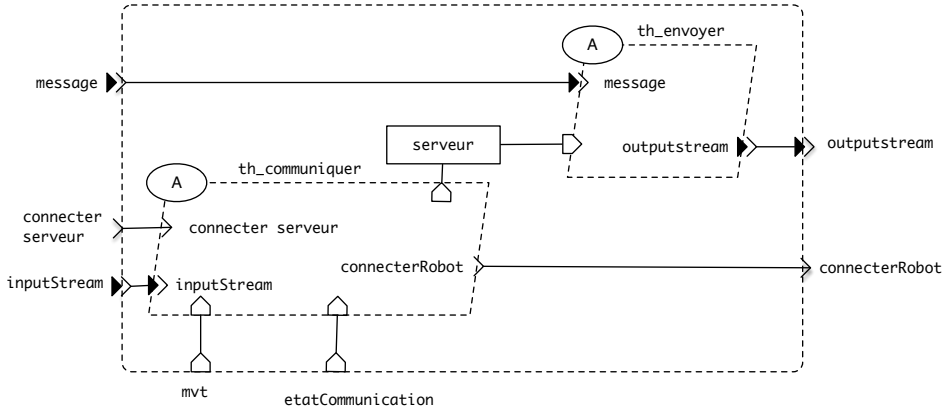

Fig. 2: Diagramme fonctionnel du groupe de threads gestion du moniteur

1.2.2 Description des threads du groupe gestion du moniteur

Remplissez le tableau ci-dessous pour expliquer le rôle de chaque thread et donner son niveau de priorité.

Tab. 1: Description des threads du groupe `th_group_gestion_moniteur`

Nom du thread	Rôle	Priorité
tCommuniquer	Prend en charge les messages entrants depuis le moniteur	25
tEnvoyer	Envoi l'ensemble des messages du superviseur au moniteur	30
...	...	...

1.2.3 Diagrammes d'activité du groupe gestion du moniteur

Décrivez le comportement de chacun de vos threads avec des diagrammes d'activité. Apportez les explications qui vous semblent nécessaires pour comprendre votre conception. A titre d'exemple les diagrammes fonctionnels tirés du document de conception sont remis.

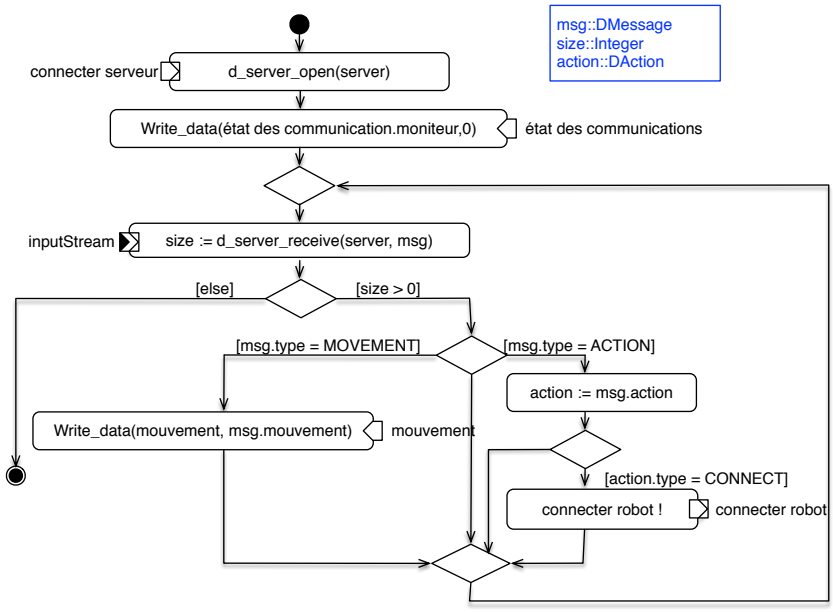

Fig. 3: Diagramme d'activité du thread `th_communiquer`

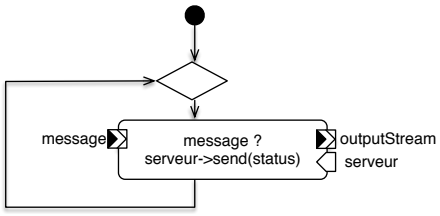

Fig. 4: Diagramme d'activité du thread `th_envoyer`

1.3 Groupe de threads gestion du robot

1.3.1 Diagramme fonctionnel du groupe gestion robot

Ajoutez le diagramme fonctionnel du groupe de threads de gestion du robot.

1.3.2 Description des threads du groupe gestion robot

Remplissez le tableau ci-dessous pour expliquer le rôle de chaque thread et donner son niveau de priorité.

Tab. 2: Description des threads du groupe `th_group_gestion_robot`

Nom du thread	Rôle	Priorité
...	...	...

1.3.3 Diagrammes d'activité du groupe robot

Décrivez le comportement de chacun de vos threads avec des diagrammes d'activité. Apportez les explications qui vous semblent nécessaires pour comprendre votre conception.

1.4 Groupe de threads vision

1.4.1 Diagramme fonctionnel du groupe vision

Ajoutez le diagramme fonctionnel du groupe de threads de vision.

1.4.2 Description des threads du groupe vision

Remplissez le tableau ci-dessous pour expliquer le rôle de chaque thread et donner son niveau de priorité.

Tab. 3: Description des threads du groupe `th_group_vision`

Nom du thread	Rôle	Priorité
...	...	...

1.4.3 Diagrammes d'activité du groupe vision

Décrivez le comportement de chacun de vos threads avec des diagrammes d'activité. Apportez les explications qui vous semblent nécessaires pour comprendre votre conception.

2 Analyse et validation de la conception

Pour les trois exigences suivantes montrer en quoi votre conception permet d'y répondre.

Exemple de ce qui est attendu

Voici un exemple avec pour exigence : « une fois la communication établie avec le robot, les ordre de mouvement sélectionnés par l'utilisateur sur le moniteur sont transmis au robot. »

Il faut justifier à travers la conception que cette exigence est bien prise en considération et estimer le temps maximum que peut prendre la transmission d'un ordre.

Pour illustrer cela, je m'appuie sur la première conception du système qui est faite dans le document « Dossier de conception ».

Justification : une fois la communication établie, les ordres de l'utilisateur sont reçus par le superviseur via le port `connecter_serveur` du thread `th_communiquer` (attente de la fonction `d_server_receive` sur la figure 4).

Le diagramme d'activité du thread (fig. 4) montre qu'à la réception d'un message de type MOUVEMENT la valeur qu'il porte est écrite dans la donnée partagée `mouvement`.

Cette donnée est lue périodiquement par le thread `th_deplacer`. Son diagramme d'activité (fig. 7) montre qu'en fonction de la valeur de la donnée `mouvement` les variables internes à `th_deplacer` nommées `gauche` et `droite` sont mises à jour pour ensuite être transmises au robot via l'appel à la fonction `d_robot_set_motor`.

Estimation du temps de traitement : la figure **??** montre la séquence d'exécution la plus désavantageuse pour la prise en compte d'un ordre de mouvement. L'utilisateur commence par sélectionner son ordre sur l'interface graphique qui ajoute un délai de 200 ms avant de le transmettre au superviseur (on ignore ici les délais réseaux). Le thread `th_communiquer` étant le plus prioritaire, il traite immédiatement le message (on remarque par exemple sur la figure la préemption du thread `th_deplacer`). Le temps de traitement de `th_communiquer` est estimer à 1 ms d'après le document « Dossier de conception ».

La pire des situations est une exécution du thread `th_deplacer` juste avant la réception du message de mouvement qui devra attendre 1 seconde avant de lire la nouvelle valeur de `mouvement`. Le temps de transmission de l'ordre au robot a été estimé à 41 ms (voir p. 12 du document « Dossier de conception »).

Nous avons donc un délai possible de l'ordre de 0.2+0.001+1+0.041 = 1.242 s entre la sélection d'un mouvement par l'utilisateur et sa réception par le robot.

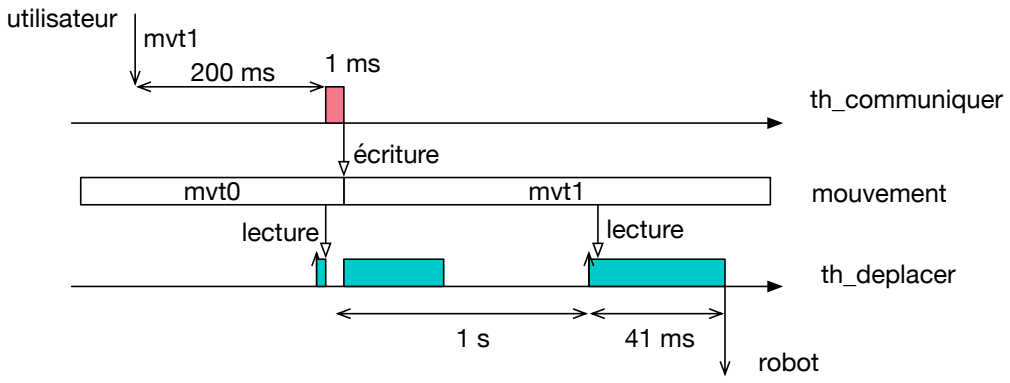

Fig. 5: Exemple de pire séquence d'exécution pour la transmission d'un ordre de mouvement au robot.

2.1 Exigence 1 : En cas de perte de communication entre le robot et le superviseur, le robot est stoppé et le nouvel état est signalé à l'utilisateur via le moniteur.

Expliquez en vous appuyant sur votre conception comment vous garantissez que cette exigence est toujours respectée.

Fournissez un chronogramme (ou un diagramme de séquence si vous préférez) avec l'ordonnancement de votre système et estimer la plus grande durée entre l'instant où la communication est perdue et l'instant où l'utilisateur en est informé.

2.2 Exigence 2 : La communication entre le robot et le superviseur est déclarée perdue si et seulement si trois échecs successifs de communication entre le robot et le superviseur surviennent.

Expliquez en vous appuyant sur votre conception comment vous pouvez garantir que cette exigence est toujours respectée.

2.3 Exigence 3 : L'image qui est affichée sur le moniteur ne doit pas être plus vieille que 600 ms (différence de temps entre la capture de l'image et son affichage sur le moniteur).

Expliquez en vous appuyant sur votre conception comment vous garantissez que cette exigence est toujours respectée.

Fournissez un chronogramme (ou un diagramme de séquence si vous préférez) avec l'ordonnancement de votre système qui montre en quoi cela répond bien à cette exigence. Expliquez.

3 Transformation AADL2XENO

Cette section est consacrée aux moyens pour passer d'un modèle AADL au code. Pour chacun des éléments AADL, vous allez expliquer **comment le traduire en C** et quels **services de Xenomai** vous avez utilisés **en expliquant ce qu'il fait**. N'hésitez pas à illustrer avec des extraits de code.

3.1 Thread

3.1.1 Instanciation et démarrage

Expliquer comment vous implémentez sous Xenomai l'instanciation et le démarrage d'un thread AADL.

Exemple de réponse : Un thread AADL est instancié en C par une tâche (`RT_TASK`) Xenomai. Pour cela, une structure `RT_TASK` est déclarée comme variable globale pour chaque tâche.

Le service `rt_task_create` est utilisé pour créer la tâche, c'est-à-dire réserver son space mémoire et la déclarer au noyau, et `rt_task_start` pour lancer l'exécution de la tâche.

3.1.2 Code à exécuter

Comment se fait le lien sous Xenomai entre le thread et le traitement à exécuter.

3.1.3 Niveau de priorités

Expliquer comment vous fixez sous Xenomai le niveau de priorité d'un thread AADL.

3.1.4 Activation périodique

Expliquer comment vous rendez périodique l'activation d'un thread AADL sous Xenomai.

3.1.5 Activation événementielle

Expliquer les moyens mis en œuvre dans l'implémentation sous Xenomai pour gérer les activations événementielles d'un thread AADL.

3.2 Port d'événement

3.2.1 Instanciation

Comment avez-vous instancié un port d'événement ?

3.2.2 Envoi d'un événement

Quels services ont été employés pour envoyer un événement ?

3.2.3 Réception d'un événement

Comment se fait la synchronisation sur événement ?

3.3 Donnée partagée

3.3.1 Instanciation

Quelle structure instancie une donnée partagée ?

3.3.2 Accès en lecture et écriture

Comment garantissez-vous sous Xenomai l'accès à une donnée partagéé ?

3.4 Ports d'événement-données

3.4.1 Instanciation

Donnez la solution retenue pour implémenter un port d'événement-données avec Xenomai.

3.4.2 Envoi d'une donnée

Quels services avez-vous employés pour envoyer des données ?

3.4.3 Réception d'une donnée

Quels services avez-vous employé pour recevoir des données ?

4 Application de la transformation AADL2Xenomai

Vous trouverez à la fin de ce document (voir annexe **??**) une application très simple modélisée en AADL. Mettez ici le code de cette application en suivant scrupuleusement vos règles.

Ne mettez pas vos fichiers d'en-tête (sauf des extraits importants si nécessaire) et structurer le plus simplement le code. L'important est de montrer la cohérence entre les règles énoncées ci-dessus et le code que vous produisez sur l'exemple.

Supprimer la partie suivante du rapport final

A Cas d'étude à traduire

L'application est un démonstration réalisée par Texas Instrument pour son système DaVinci. Ce système est dédié aux applications multimédia (vidéo, audio, etc.) et est fourni avec un BSP basé sur Linux. Il équipe des audiophones, des systèmes de surveillance, des équipements médicaux, etc.

Le diagramme **??** représente le système DaVinci et l'ensemble des éléments avec lesquels il est connecté.

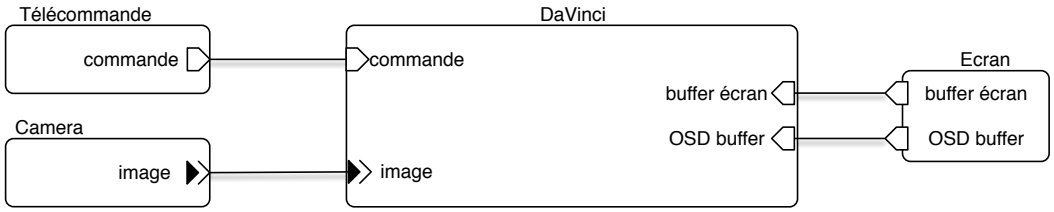

Fig. 6: Vue générale du système

Les différents éléments du diagramme sont :

— Une télécommande infrarouge qui écrit dans la zone mémoire tampon `commande` les commandes émises. Cette zone est accessible en lecture depuis l'application.
— Une caméra qui produit des données et un signal `image` quand une image est disponible.
— Un écran pour afficher les images capturées et les informations sur le système. L'affichage est mis à jour en écrivant dans le buffer de l'écran (`buffer écran`). Les données contenues dans `OSD buffer` sont des informations permettant d'afficher les menus à l'écran.

A.1 Description de l'application de démonstration

Le diagramme **??** fournit la description de l'application de démonstration de décodage vidéo de Texas Instrument.

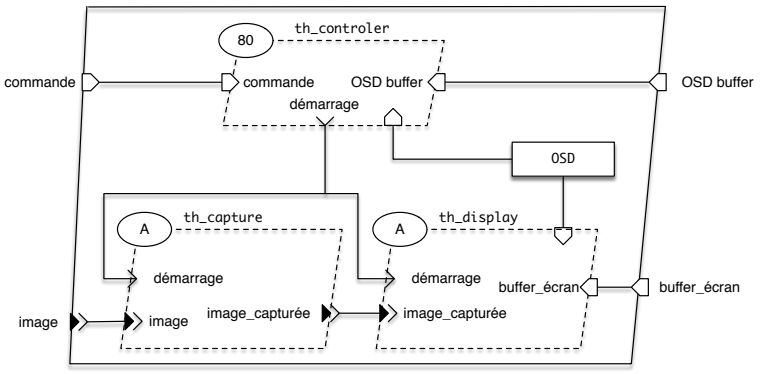

Fig. 7: Diagramme fonctionnel de l'application

A.1.1 Description du thread th_controler

Le thread `th_controler` (figure **??**) initialise le système, synchronise toutes les tâches, puis lit les informations en provenance de la télécommande et met à jour le menu sur l'écran ainsi que les données de configuration représentées par `OSD`.
Pour cela deux fonctions sont utilisées : `initialiser_le_système` et `calcul_osd`.

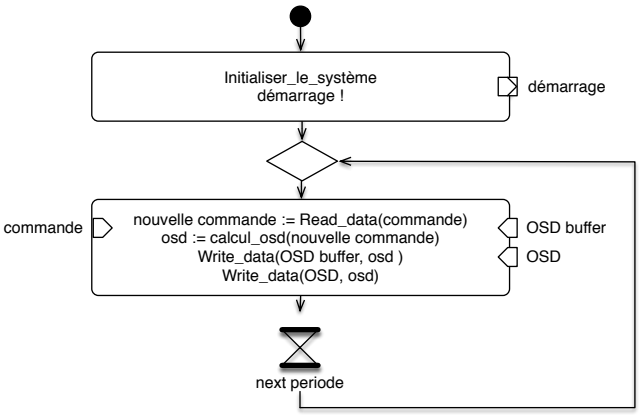

Fig. 8: Diagramme d'activité du thread th_controler

A.1.2 Description du thread th_capture

Le thread `th_capture` (figure **??**) fait l'acquisition d'une image quand elle est disponible et la met dans une file de message.
La fonction `acquistion_image` est une fonction bloquante sur le port `image` et retourne la valeur du port quand l'évènement se produit. En conséquence, vous n'avez pas à utiliser de service pour gérer le port de donnée-événement `image`, la fonction `acquistion_image` fait le travail.

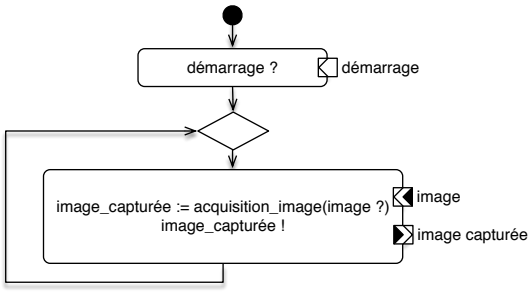

Fig. 9: Diagramme d'activité du thread th_capture

A.1.3 Description du thread th_display

Le thread th_display (figure **??**) met à jour le buffer de l'écran en fonction des paramètres contenu dans la données OSD. Pour cela la fonction `encode` est utilisée.

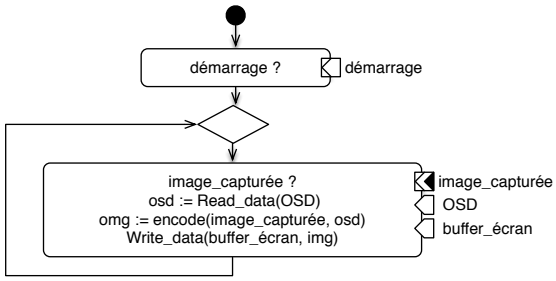

Fig. 10: Diagramme d'activité du thread th_display

A.2 Priorités

Les priorités des threads sont fixées (plus la priorité est élevée plus la tâche est prioritaire) :

Thread	Priorité
`th_controler`	60
`th_capture`	80
`th_display`	70