Université de Bretagne Occidentale

Master 2 Informatique

Département Informatique

2020/2021

Vérification, Fiabilité et Sécurité

Modélisation et prototypage avec AADLInspector/Ocarina d'un Système Embarqué

Auteur :
William PENSEC

Auteur :
Timothé LANNUZEL

11 février 2021

Sommaire

I Introduction

L'objectif de ce projet est de construire un modèle AADL à partir d'un système proposé dans une publication scientifique puis de tester le modèle sur AADLInspector et d'en faire une simulation avec l'outil Ocarina. La version utilisée pour AADLInspector est la version de démonstration sur Windows 10 et Ocarina est utilisé sur une machine virtuelle sur Linux (Ubuntu 16.04).

Nous avons choisi l'article : "Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles" de MM et Mme BENJAMIN HOROWITZ, JUDITH LIEBMAN, CEDRIC MA, T. JOHN KOO, ALBERTO SANGIOVANNI - VINCENTELLI et S. SHANKAR SASTRY. Ce document parle des drones autonomes et de leurs systèmes de navigation.

Nous avons choisi de concevoir plusieurs versions du modèle afin de mieux montrer le système sous différentes possibilités d'interprétation et d'implémentation.

II Description du système étudié

Afin d'étudier le système, nous avons mis en place la solution suivante (voir Figure 1). Comme on peut le voir sur l'image [1] et comme il est dit dans l'article, nous avons pris pour `THREADS` principaux `Fusion` et `Control` qui sont appellés respectivement `dataProc` et `controller` dans notre code. Ainsi, le thread `Fusion` va recevoir les valeurs du `GPS` et de l'`INS`, puis va les traiter et ensuite les envoyer au thread `Control` afin de transformer ces valeurs du `GPS` et de l'`INS` en une donnée compréhensible pour les servos servant à diriger l'appareil.

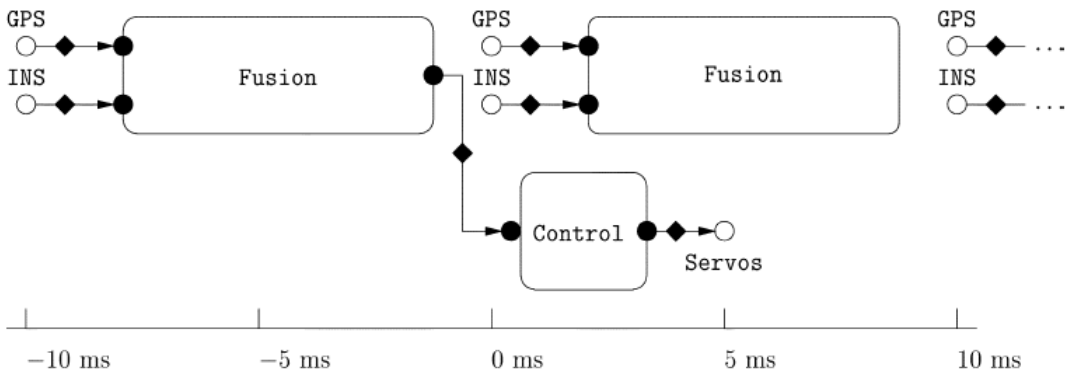

FIGURE 1 – Système appliqué à notre solution

1. L'image présentée ici vient de l'article étudié et correspond à la Fig. 7 Refined Giotto program

Nous avons, à partir de ce système, conçu 3 versions différentes qui seront détaillées dans la partie suivante. Chaque version correspond à une implémentation possible entre ce qui est mis dans l'article ou ce qui est apparu dans notre conception du système.

III Description et explication du modèle AADL

Nous avons tenu à faire différentes versions du système car dans l'article ils mentionnaient plusieurs solutions et certains points n'étaient pas très clair. Par exemple, on peut citer les données manquantes telles que le temps de récupération des données venant du `GPS` et de l'`INS`. Sinon les données de la période du `Data Processor` et du `Time Based Controller` sont assez clair et il était expliqué dans l'article pourquoi ils prennaient les valeurs choisies.

Une trace d'exécution sur AADLInspector est trouvable à la fin du fichier dans la partie V à la page 8.

III.1 Version 1

La première version est une version annexe avec 1 `CPU` et 5 threads. Ces threads sont :

- Un thread pour la gestion de l'envoi et de la réception des données (`dataProc`)
- Un deuxième afin de formatter les données afin que les servos moteurs comprennent l'information et ce qu'ils doivent exécuter (`controller`).
- Un troisième qui gère le `GPS` (`device_GPS`).
- Un quatrième qui gère l'`INS` (`device_INS`).
- Un cinquième qui gère les `Servos` (`device_SERVOS`).

Dans la figure 2, les cadres en pointillés représentent les différents threads du système. Les threads des appareils physiques ne permettent que de communiquer entre l'appareil physique et le thread `dataProc` afin de recevoir des données.

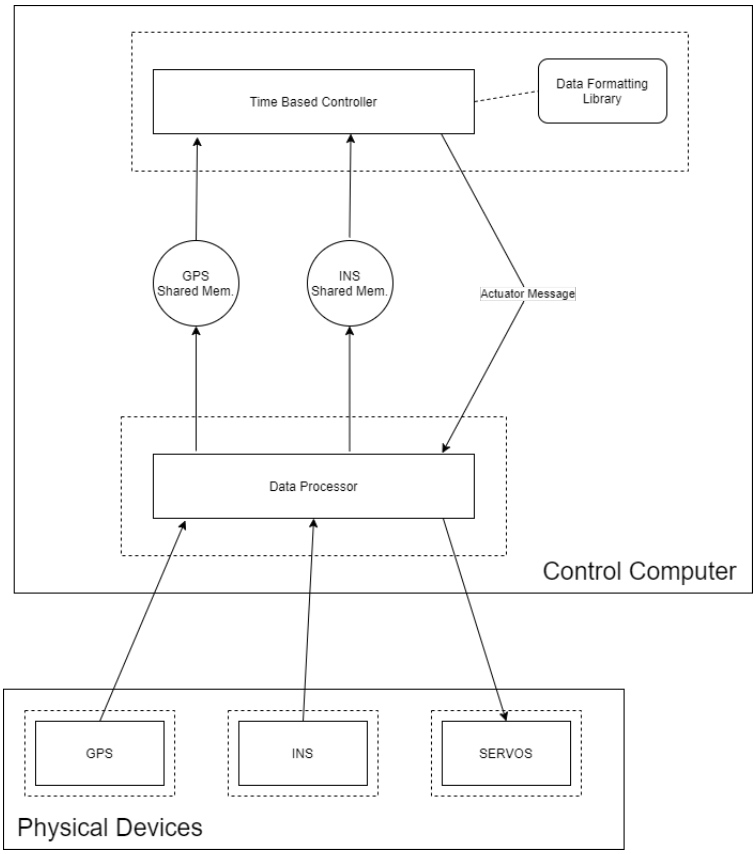

FIGURE 2 – Système appliqué avec 1 CPU et 5 threads

Nous présentons à présent les différentes valeurs appliquées pour chacun des threads. Le fait de mettre un décalage seulement au `device_Servos` permet qu'il démarre seulement après que le `dataProc` ait lu et envoyé la valeur du `controller` aux servos sinon il s'exécuterait pour rien et entrainerait donc une petite perte de performance. Pour calculer le décalage on prend le temps d'exécution du `dataProc` + celui du `controller`.

	Ordonnancement	Période	Deadline	Temps d'exécution	Priorité	Décalage
dataProc	Périodique	20 ms	20 ms	0 à 10 ms	9	
controller	Périodique	20 ms	20 ms	0 à 5 ms	8	
device_GPS	Périodique	20 ms	20 ms	0 à 1 ms	8	
device_INS	Périodique	20 ms	20 ms	0 à 1 ms	8	
device_Servos	Périodique	20 ms	20 ms	0 à 1 ms	8	15 ms

III.2 Version 2

La deuxième version implémentée est une version avec 1 `CPU` qui prend en charge 2 threads :

- Un thread pour la gestion de l'envoi et de la réception des données (`dataProc`).
- Un deuxième afin de formatter les données afin que les servos moteurs comprennent l'information et ce qu'ils doivent exécuter (`controller`).

L'image suivante[2] montre le schéma d'architecture de cette version. Le `CPU` correspond au `Control Computer`.

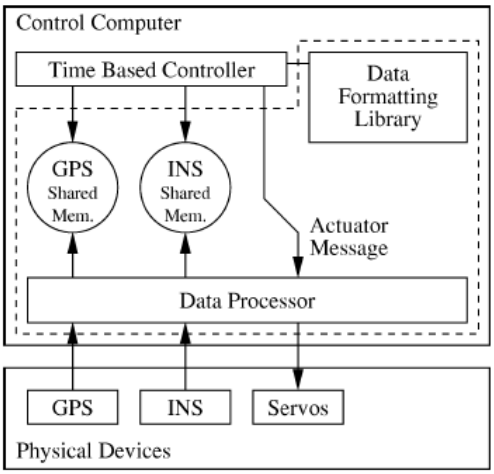

FIGURE 3 – Système appliqué avec 1 CPU

Nous présentons à présent les différentes valeurs appliquées pour chacun des threads.

	Ordonnancement	Période	Deadline	Temps d'exécution	Priorité
dataProc	Périodique	20 ms	20 ms	0 à 10 ms	9
controller	Périodique	20 ms	20 ms	0 à 5 ms	8

Actuellement, nous avons seulement implémenter cette version avec Ocarina et tout est fonctionnel. Nous avons implémenté cette version afin de montrer qu'il est possible d'envoyer des données et d'en recevoir avec une exécution correcte selon les périodes et temps d'exécution des tâches. Voici un exemple d'exécution du code une fois compilé avec Ocarina :

2. L'image présentée ici vient de l'article étudié et correspond à la Fig. 8 First implementation of UAV platform. The dashed line encloses the platform implementation.

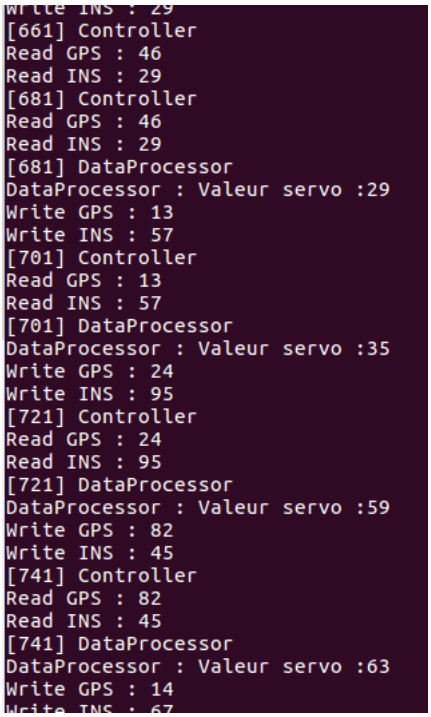

FIGURE 4 – Exécution de la Version 2

III.3 Version 3

La deuxième version implémentée est une version avec 2 `CPU` qui prennent chacun en charge 1 thread.

- Le premier `CPU` correspond au `Control Computer` qui permet de traiter les informations reçues du deuxième `CPU` avec les données du `GPS` et de l'`INS` ; puis ensuite permet de renvoyer la valeur calculée au deuxième `CPU`.
- Le deuxième `CPU` correspond au `Data Computer` qui permet de récupérer les informations envoyées par le `GPS` et par l'`INS`. Puis envoyer ces valeurs au premier `CPU` qui va donc calculer puis ensuite une fois la valeur reçu du premier `CPU`, le `Data Computer` appelle la partie `Data Formatting Library` qui va rendre compréhensible la valeur pour les servos moteurs de l'appareil ; et enfin, envoi cette valeur formatée aux servos.

L'image suivante[3] montre le schéma d'architecture de cette version. Le `CPU1` correspond au `Control Computer`, le `CPU2` correspond au `Data Computer` et les `Physical Devices` communiquent avec le `CPU2`.

3. L'image présentée ici vient de l'article étudié et correspond à la Fig. 9 Second implementation of UAV platform. The dashed line encloses the platform implementation.

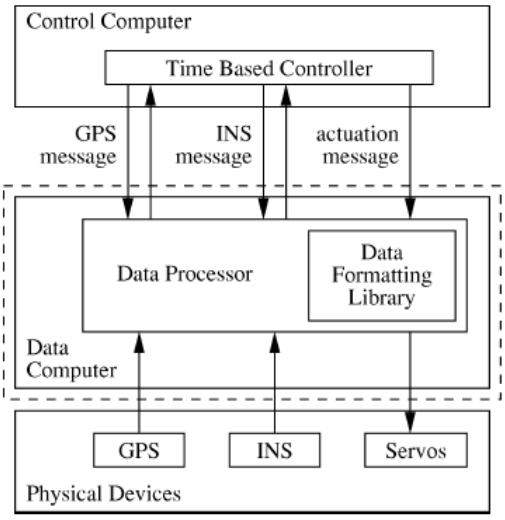

FIGURE 5 – Système appliqué avec 2 CPU

Nous présentons à présent les différentes valeurs appliquées pour chacun des threads. Le fait de mettre un décalage seulement au controller permet qu'il démarre seulement après que le `dataProc` ait lu une valeur du GPS et de l'INS sinon il s'exécuterait pour rien et entrainerait donc une petite perte de performance. Pour calculer le décalage on prend le temps d'exécution du `dataProc` et c'est tout.

	Ordonnancement	Période	Deadline	Temps d'exécution	Priorité	Décalage
dataProc	Périodique	20 ms	20 ms	0 à 10 ms	9	
controller	Périodique	20 ms	20 ms	0 à 5 ms	8	10 ms

IV Utilisations potentielles du modèle

Les vérifications que les modèles permettent de faire sont de simuler le système de navigation d'un véhicule autonome (type UAV) selon les données reçues des capteurs présents sur l'appareil comme par exemple le `GPS` et l'`INS`. Ces données, une fois traitées, permettent d'actionner ou non les servos moteurs de l'appareil afin de recalibrer la direction du drone. Les vérifications qui sont intéressantes et nécessaires de faire sont de vérifier que le modèle respecte bien les deadlines des tâches, en évitant que la somme des périodes des tâches soient supérieur à la capacité totale du CPU utilisé.

Les paramètres ou données intéressants à déterminer pour le système étudié sont les périodes, deadline, temps d'exécution et ordre d'exécution des tâches (`dataProc`

et `controller`) afin de garantir un bon fonctionnement du système et éviter les malfonctionnements.

V Annexes : traces sur AADLInspector

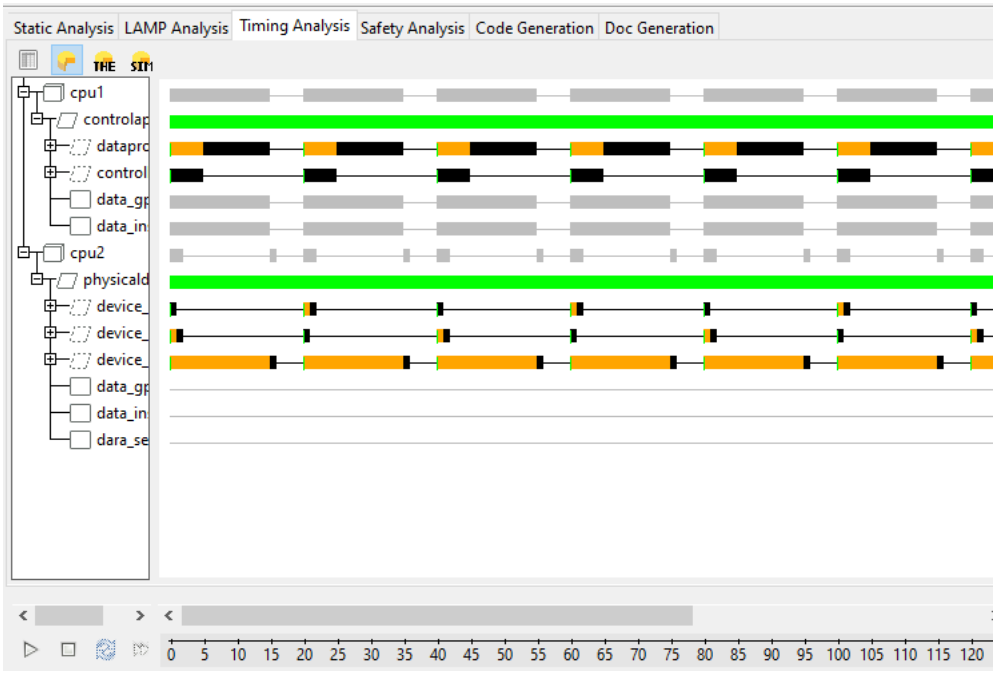

FIGURE 6 – Trace d'exécution de la Version 1 du système

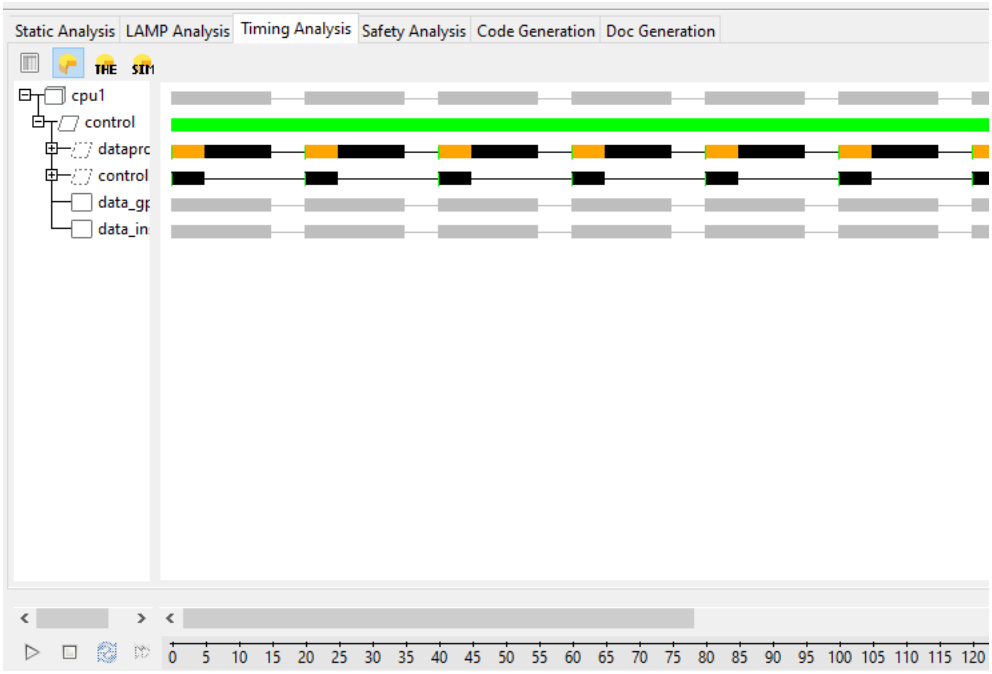

FIGURE 7 – Trace d'exécution de la Version 2 du système

FIGURE 8 – Trace d'exécution de la Version 3 du système