**Systémes Temps Réel Embarqués Critiques**

---

**TP AADL Ordonnancement**

# Introduction

Dans ce TP, nous allons nous intéresser à la modélisation d'architecture en AADL, avec pour objectif de réaliser des analyses d'ordonnancement. Nous limitons cette étude à l'analyse de systèmes ARINC653 bien que AADL puisse être utilisé pour modéliser bien d'autres méthodes d'ordonnancement.

# Exemple de référence

L'objectif de cette partie du TP est simplement de manipuler les outils d'édition et d'analyse de modèles AADL avant de traiter les exercices qui suivent. Nous fournissons pour cela une archive qui contient un exemple de référence. Cet exemple représente la même configuration que celle de l'exemple de référence utilisé lors du TP ARINC653. Nous rappelons ici la description de cet exemple.

## Description de l'exemple

L'exemple que nous proposons est un système constitué de 4 partitions : part1, part2, part3 et part4. Chaque partition exécute une tâche périodique de période égale à 1 seconde. Le partitions sont ordonnancés cycliquement avec un « Major Frame » de 1 seconde et chaque partition se voit alloué successivement une « Partition Window » de 250 millisecondes.

Les communications entre partitions se font via des port de type « sampling ». Les partitions part1 et part2 ont chacune un port de sortie, nommés respectivement p1 pour part1 et p2 pour part2. La partitions part3 a deux ports d'entrée p3 et p4, et la partition part4 a un port d'entrée p5.

Dans une partition avec un port de sortie, le code de la tâche périodique exécutée toutes les secondes est simple : il affiche la valeur qu'il s'apprête à envoie sur le port, puis l'envoie. Dans une partition avec port(s) d'entré, le code de la tâche périodique exécutée toutes les secondes est simple : il lit puis affiche la (ou les données) reçue(s) sur le(s) port(s).

Les « channel » de communication permettent aux données d'être transmises depuis un port de sortie vers un port d'entrée. Dans notre exemple, le port p1 est connecté au port p3, le port p2 est connecté aux ports p4 et p5.

Enfin, nous avons définit que chaque partition se voit alloué un espace mémoire de 150000 octets.

Le schéma ci dessous résume l'architecture considérée dans cet exemple.

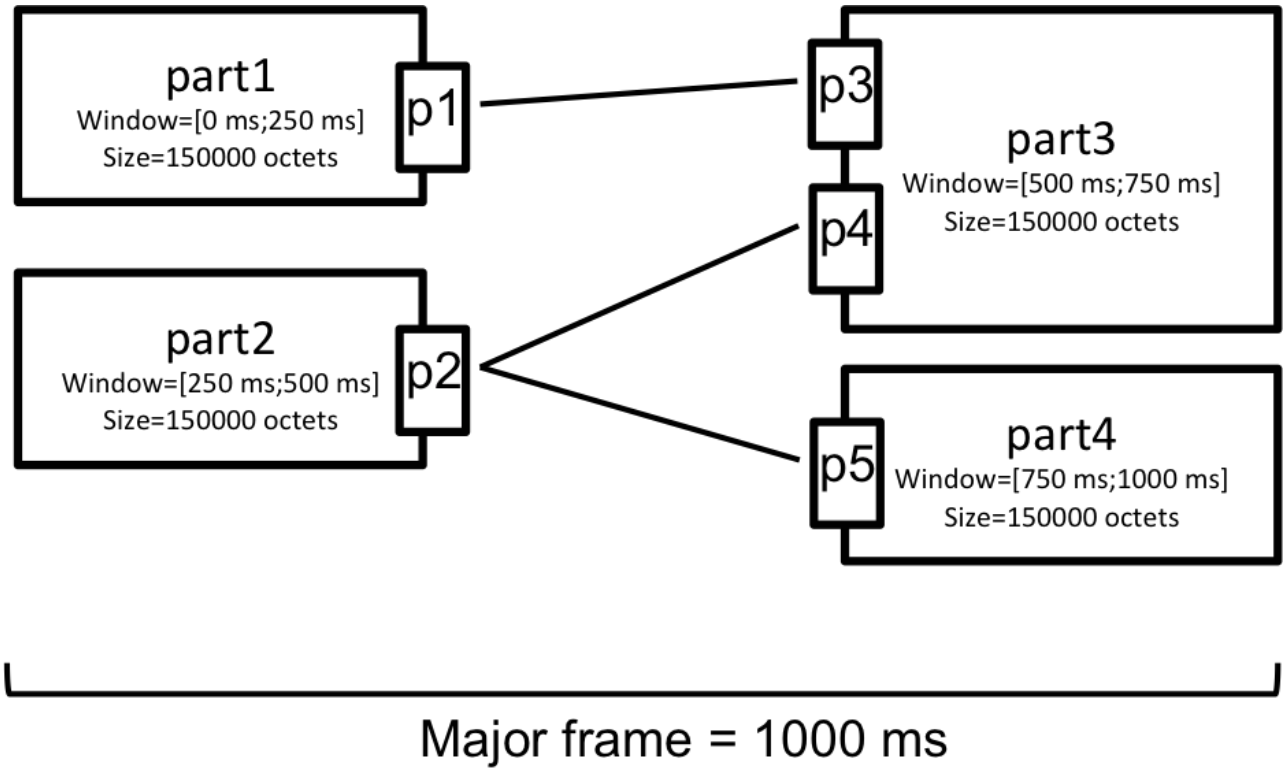



FIGURE 1

En comparant le modèle AADL avec la description XML de la configuration du système que nous avions fourni pour le TP ARINC653, nous constatons que le modèle AADL étend la description XML avec la modélisation du contenu des partitions en termes de tâches (process ARINC653), avec leurs propriétés temporelles et leur accès à des variables partagées. Ainsi, le code de ces tâche peut-être (partiellement ou intégralement) généré, et leur pire temps de réponse peut-être analysé à partir du modèle d'architecture.

## Description de l'archive

L'archive qui se trouve ici contient le modèle AADL correspondant à ce système.

L'archive contient 3 fichiers :

1. aadl contient la définition des composants logiciels de l'architecture
2. aadl contient la définition des composants logiciels de l'architecture
3. aadl contient le système racine à partir duquel sera construite l'arborescence de composants.

### Composants logiciels

Ce fichier contient la définition de 4 implémentations de composants processus: part1.impl, part2.impl, part3.impl et part4.impl. Chaque implémentation de composant contient la même structure de sous-composant par héritage de partition_sw.impl. Cependant, les types de composant processus part1, part2, part3 et part4 diffèrent car ils n'ont pas exactement les mêmes interfaces.

L'implémentation de composant processus partition_sw.impl contient un unique thread periodic_1000. Ce thread est périodic, de période 1000 millisecondes. Son comportement est décrit avec l'annexe comportementale AADL : lorsque le thread se réveil (transition avec la condition « on dispatch »), il fait un calcul de 100 millisecondes (computations (100ms)) puis se rendort (atteint l'état « complete » s1).

### Composants matériels

Ce fichier contient la définition de la plate-forme d'exécution. Ici, la description est simple: une implémentation de composant processeur est constitué de quatre processeurs virtuels. Chaque processeur virtuel représente une partie de la ressource de calcul (un peu comme une machine virtuelle).

La politiques d'ordonnancement associée au processeur est ARINC653. La « Major frame » est renseignée via la propriété « Module_Major_Frame » du « property set » ARINC653.

Les « partition windows » sont définies via la propriété « Module_Schedule » : de 0 à 250 ms, la ressource de calcul principale est attribuée à la ressource virtuelle vp1, de 250 à 500 ms, la ressource de calcul principale est attribuée à la ressource virtuelle vp2, etc.

La politique d'ordonnancement associée à chaque ressource de calcul virtuelle est RMS (voir la propriété « Scheduling_Protocol » dans le type de « virtual processor » vp_rms).

### Arborescence de composants

L'intégration des composants matériels et logiciels est fournie dans le fichier integration.aadl. Le système racine (appelé root_system.impl) contient un processeur et quatre partitions.

Les connections entre ports des partitions sont définies.

Enfin, la propriété « Actual_Processor_Binding » est utilisée pour associer chaque partition logique (processus AADL) à une partition physique (virtual processor AADL).

# Première manipulation d'outils avec l'exemple de référence

## Lancement de l'outil

L'environnement utilisé pendant de TP est OSATE (Open-Source AADL Tool Suit Environment). Pour automatiser les analyses de temps de latence, nous utiliserons un plugin de l'outil OSATE. OSATE est développé au Software Engineering Institute (USA). Pour automatiser les analyses de temps de réponse des tâches, nous utiliserons l'outil AADL Inspector développé par la société Ellidiss.

Lancez OSATE depuis le terminal:

```
source /infres/s3/borde/Install/env_osate
/infres/s3/borde/Install/osate2/osate &
```

Choisissez un workspace à l'emplacement de votre choix (conseil: créez un répertoire dédié à un emplacement dont vous vous rappellerez, par ex: $(HOME)/studies/se301/osate_workspace).

Comme vous pouvez le voir, il s'agit d'un environnement basé sur Eclipse. De nombreuses fonctionnalités sont ainsi disponible (Ctrl+click, ou Ctrl+space par exemple).

## Import de l'exemple

Avec cet exemple, nous allons commencer par une petite présentation des outils:

- Téléchargez l'archive qui se trouve ici, et désarchivez la.
- [telecom-paristech.fr/~borde/aadl/tp/reference-example.tar.gz]
- Importez le projet: cliquez sur « *File –> Import ...*« .
- Dépliez le répertoire « *General*» puis sélectionnez « *Existing projects in Workspace*« , cliquez sur « *next*« .
- Sélectionnez le répertoire résultant de l'étape 1 (champ « *Select root directory:*« ).
- Sélectionnez le projet « *reference-example*« .
- Cliquez sur Finish.

## Visualisation du modèle

Après réalisation des actions décrites ci-dessus, vous avez obtenu un projet « *reference-example* » qui contient plusieurs fichiers AADL: *hadware.aadl, software.aadl, et integration.aadl.*

Ouvrez ces modèles AADL, et consulter leur contenu pour comprendre l'architecture décrite. Identifiez en particulier les composants et propriétés utiles aux analyses temporelles vues en cours.

Pour vous aider à visualiser le contenu du fichier, vous pouvez

- utiliser la vue « *Outline*« , sur la droite de l'environnement OSATE;
- utiliser la vue graphique:
  - sélectionnez un composant (soit dans la vu « Outline », soit en sélectionnant son nom complet)
  - clique droit
  - cliquez sur Open Diagram; la représentation graphique du composant sélectionné apparait. Pour information, cette vue graphique est synchronisée avec le contenu du modèle textuel et peut aussi servir d'éditeur graphique (vous pouvez modifier le modèle à partir de cette visualisation graphique).

## Instanciation du modèle

Dans OSATE, les plugins d'analyse s'appuie sur une représentation plus synthétique du modèle qui s'appelle le modèle d'instance. Par analogie avec les langages orientés objets, nous « pourrions considérer » un type de composant AADL comme une « interface », une implémentation de composant AADL comme une « classe », et une instance de composant comme un « objet ».

En AADL, les instances sont construites à partir d'un système racines: en prenant les sous-composants de ce système racine, puis leurs sous-composants, on obtient un arbre de composants et chaque noeud de l'arbre correspond à une instance.

Pour instancier un modèle AADL, faites un click droit sur un « *system implementation* » (appelé système racine de l'instanciation) dans le « *outline view* » (compartiment de droite de Eclipse), puis cliquez sur « *Instantiate system*« . Dans votre projet, un répertoire « *instances* » a été créé, il contient le modèle d'instance correspondant à l'arborescence de composants contenus dans le système racine que vous avez sélectionné.

ATTENTION: dans OSATE, il faudra refaire cette phase d'instantiation chaque fois que vous modifiez le modèle et que vous voulez relancer une nouvelle analyse.

## Analyse de temps de réponse des tâches

Etapes à suivre pour connaitre le pire temps de réponse des tâches:1. instanciez le système *root_system.impl* 2.

cela produit, dans un répertoire *instances*, un nouveau fichier qui correspond au modèle d'instance3. sélectionnez le fichier produit4. cliquez sur l'onglet RAMSES (tout en haut de l'environnement), puis cliquez sur « Launch AADL Inspector »5. cela lance une nouvelle fenêtre5.1. sélectionnez le chemin vers l'outil AADL Inspector: /infres /s3/borde/Install/AADLInspector5.2. sélectionnez le mode Graphical Interface6. cliquez sur OK

Vous arrivez dans l'outil AADL Inspector, et votre modèle a été ouvert. Quelques informations utiles pour utiliser l'outil:

- pour connaître le pire temps de réponse des threads, cliquez sur l'onglet « Schedulability » dans la fenêtre en haut à droite de l'outil; cliquez sur la petite icone « *SIM* » (pour simulation-based schedulability test); dépliez la partie « *worst case response time*«
- pour visualiser le chronogramme d'ordonancement, cliquez sur le petit fromage à gauche de « *THE*« (lui même à gauche de « SIM ») .

# Système à modéliser

Le premier exercice de ce TP consiste à représenter l'architecture de l'exemple étudié pendant le TP ARINC653. Nous rappelons ici la description de cet exemple :

La société pour laquelle vous travaillez aujourd'hui souhaite faire réaliser 4 applications par des fournisseurs différents et les intégrer sur un unique calculateur sur lequel est déployé un système partitionné ARINC653.

Ces applications sont appelées:

- Navigation,
- Radar,
- Logging,
- Display

Voici la configuration des tâches de chaque application, telle que négociée avec vos fournisseurs :

- Navigation : 2 tâches, de période 50 ms et 300 ms.

Temps d'exécution de la tâche cadencée à 50 ms : 3 ms; temps d'exécution de la tâche cadencée à 300 ms : 10 ms ;

- Radar : une tâche de période 100 ms.

Temps d'exécution : 15 ms

- Logging : une tâche de période 300 ms.

Temps d'exécution : 80 ms

- Display : deux tâches, de période 100 ms, 300 ms.

Temps d'exécution de la tâche cadencée à 100 ms : 21; temps d'exécution de la tâche cadencée à 300 ms : 39

Nous supposons que l'application Radar communique avec l'application Navigation, qui communique avec l'application Display. Cette chaine de communication met à jour l'affichage produit par le composant Display. Ces communications se feront via des ports de type « sampling » car chaque application n'a besoin de la dernière valeur produite pour fonctionner correctement. Pour simplifier, nous supposons que les applications s'échangent

de simples entiers.

Nous donnons ici la configuration temporelle au sens ARIN653 du terme:

La « major frame » du système est de 100 ms

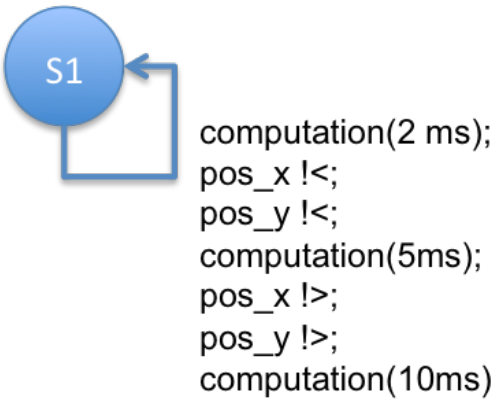
Les « partition windows » sont attribuées comme suit :

```
[0 ms .. 5 ms] --> navigation
[5 ms .. 15 ms] --> radar
[15 ms .. 30 ms] --> display
[30 ms .. 50 ms] --> logging
[50 ms .. 65 ms] --> radar
[65 ms .. 70 ms] --> navigation
[70 ms .. 90 ms] --> display
[90 ms .. 100 ms] --> logging
```

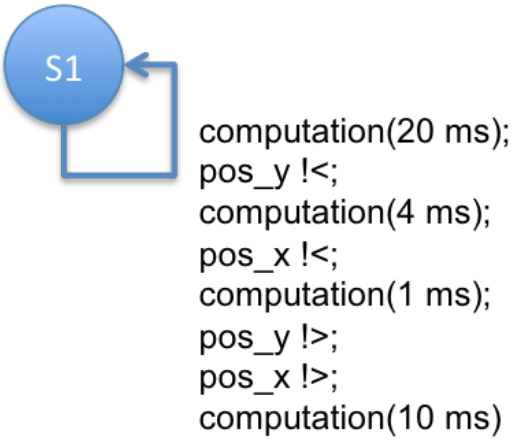
**Exercice 1:** importez l'archive fourni ici; en vous inspirant du modèle de référence, complétez l'architecture du système considéré.

# Utilisation de verrous

En regardant plus précisément l'implémentation des tâches de l'application display, on s'aperçoit que ces tâches partagent une donnée de type position. De plus, le comportement de la tâche à 100 ms est le suivant :



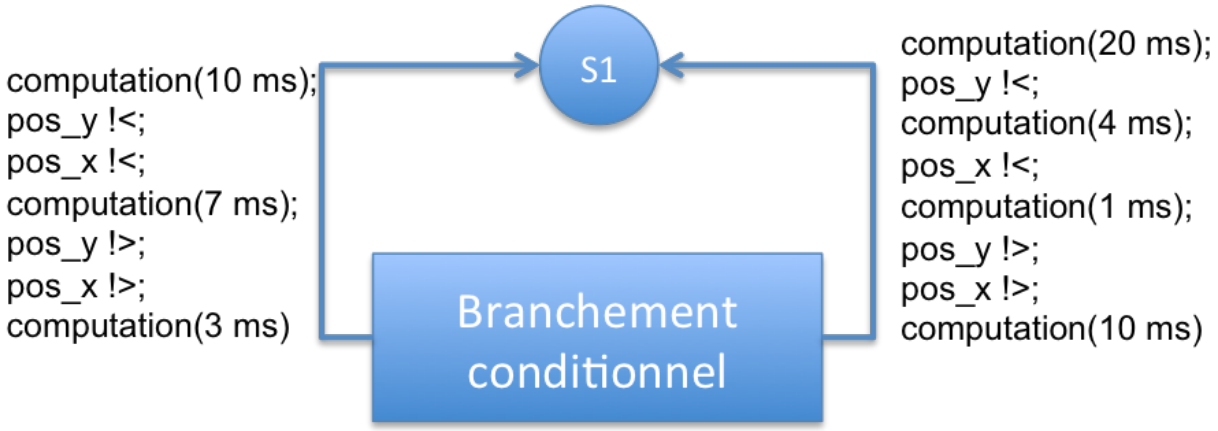

Enfin, le comportement de la tâche à 300 ms est le suivant :



**Exercice 2 :** complétez le modèle AADL. Analysez le temps de réponse des tâches. Que constatez-vous? Comment résoudre ce problème?

# Temps de blocage

Dans une nouvelle version de l'application display, la tâche à 300 ms est implémentée comme suit:




Les analyses d'ordonnancement vues en cours s'appuient sur un modèle de tâches sans branchements conditionnels. AADL Inspector analyse le même type de modèles que celui vu en cours (pas de branchement conditionnel).

**Exercice 3 :** proposez un unique lot de tâches qui permette de garantir que, s'il est ordonnançable, le lot de tâche initial est ordonnançable. Représentez ce système en AADL, et analysez le. Pouvez-vous conclure que le lot de tâche initial est ordonnançable ? Non-ordonnançable ?

**Exercice 4 :** proposez 2 deux lots de tâches qui permettent de garantir que, s'ils sont tous deux ordonnançables, le lot de tâche initial est ordonnançable. Représentez ces lots de tâches en AADL, et analysez les. Pouvez-vous conclure que le lot de tâche initial est ordonnançable ? Non-ordonnançable ?

La solution des exercices est disponible ici.

---