Chapitre I.

2ère partie

Langage de Description Et d'Analyse d'Architectures (AADL)

1. Introduction

Les langages de description d'architecture (Architecture Description Language, ADL), sont des langages propres à des domaines particuliers. Un ADL est un langage qui permet la modélisation d'une architecture conceptuelle d'un système logiciel et matériel. Il fournit une syntaxe concrète et une structure générique (framework) conceptuelle pour caractériser les architectures

Parmi les ADL, le langage d'analyse et de description d'architectures (Architecture Analysis and Design Language, AADL) qui a fait l'objet d'un intérêt croissant dans l'industrie des systèmes embarqués critiques. AADL a été standardisé par la SAE (International Society of Automotive Engineers) en 2004, pour faciliter la conception et l'analyse de systèmes complexes, critiques et temps réel dans des domaines comme l'avionique, l'automobile et le spatial. AADL fournit une notation textuelle et graphique standardisée pour décrire des architectures matérielles et logicielles. Cependant, il n'offre pas une sémantique précise et manque d'outils de vérification robuste puisque les outils utilisant ce langage sont encore pour la plupart au stade de développement et n'implémentent pas tous les éléments spécifiés dans la norme AADL [AADb,2].

2. Principes du langage

AADL peut être exprimé avec différentes syntaxes. La norme est définie de trois manières : texte brut, XML et représentation graphique (Figure 3.1). Grâce à cette multiplicité de syntaxes possibles, AADL peut être utilisé par de nombreux outils différents, graphiques ou non. Le développement de profils pour UML permet d'incorporer AADL dans les outils de modélisation UML.

AADL a été créé avec le souci de faciliter l'interopérabilité des différents outils, c'est pourquoi la syntaxe de référence est textuelle. La représentation XML permet de faciliter la création de parseurs (analyseur syntaxique) pour des applications existantes. La notation graphique se pose en complément de la notation textuelle, pour faciliter la description des architectures. Elle permet une représentation beaucoup plus claire que le

texte ou XML, mais elle est moins expressive. En tant que langage de description d'architecture, AADL permet de décrire des composants connectés entre eux pour former une architecture. Une description AADL consiste en un ensemble de déclarations de composants. Ces déclarations peuvent être instanciées pour former la modélisation d'une architecture [AADb].

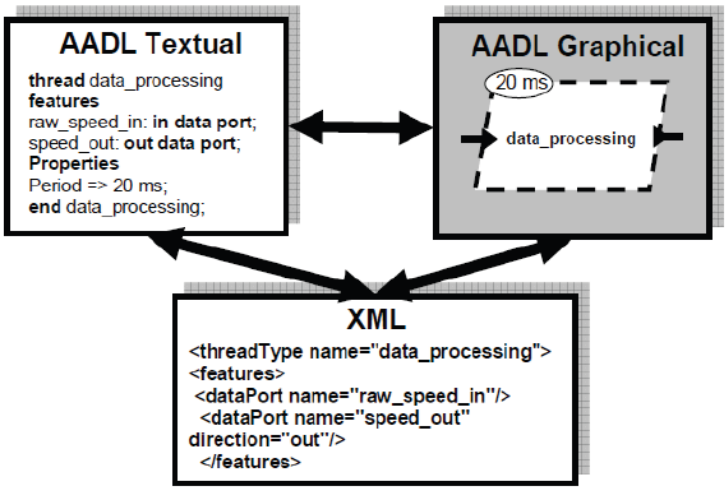

Figure 2.1 : Les représentations AADL[2]

3. *Définition des composants*

3.1.1 Les composants AADL sont définis en deux parties : l'interface et les implémentations. Un composant AADL possède une interface (type de composant) qui correspond à une, plusieurs ou aucune implémentations (implémentation de composant). Toutes les implémentations d'un composant partagent la même interface.

3.2 Les différentes catégories de composants

Les composants constituent le vocabulaire de modélisation central de l'AADL. Les composants ont une identité unique (nom) et sont déclarés comme type et implémentation dans une catégorie de composants particulière. Une catégorie de composant définit l'essence d'exécution d'un composant. Il existe trois ensembles distincts de catégories de composants :

3.2.1 Les composants logiciels **(Software Components)**

Les composants logiciels AADL sont : les données, le sous-programme, le thread, le groupe de threads et le processus, et leurs représentations graphiques sont visibles dans la Figure 2.2.

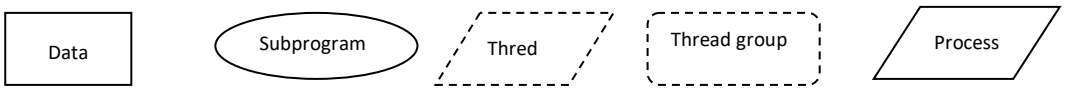

Figure 2.2 : Les composants logiciels d'AADL

3.2.1.1 Les données (Data) : Représente toutes les structures de données transmises, stockées ou partagées dans une spécification AADL. Ils peuvent avoir d'autres données comme sous-composants et des sous-programmes comme fonctionnalités. Dans ce cas, il est interprété que les sous-programmes liront et/ou modifieront les données.

Exemple :

```
data Voiture
end Voiture;

data implementation Voiture. impl
    subcomponents
        Maque: data s t r i n g;
        Couleur : data s t r i n g ;
        Serie: data i n t e g e r;
end Voiure. impl ;
```

3.2.1.2 Les sous programmes (Subprogram): Point d'entrée d'exécution dans le code source. Un type de sous-programme peut avoir des paramètres (données ou données d'événement) comme fonctionnalités entrantes, et des paramètres (données), des événements et des ports de données d'événement comme fonctionnalités sortantes. L'implémentation d'un sous-programme peut appeler d'autres sous-programmes

Exemple :

```
subprogram Fonction
    features
        X: in parameter i n t e g e r;
        Y: in parameter i n t e g e r;
        Z : out parameter i n t e g e r;
end Foncton :
```

3.2.1.3 **Thread:** Représente une unité planifiable qui exécute des instructions dans l'espace virtuel attribué au processus qui la contient. Une spécification de type de thread peut avoir, comme fonctionnalités, des ports, des accès aux données et des sous-programmes. Dans ce cas, les fonctionnalités du sous-programme représentent des sous-programmes qui peuvent être appelés à distance par des

threads exécutés sur différents processeurs. Les implémentations de threads peuvent contenir des sous-composants de données et des appels de sous-programmes.

3.2.1.4 **Thread Group:** Entité qui regroupe les threads et autres groupes de threads afin de fournir une meilleure organisation des threads dans une spécification AADL.

Exemple

```
thread Calcul
      features
            X: in data port;
            Z : out data port;
            Properties
            Dispatch_protocol=>Periodic;
            Period=> 20ms
end Calcul ;
```

Process: Représente un espace virtuel où les threads sont exécutés. Un type de processus peut avoir les mêmes fonctionnalités que celles vues dans les threads et ses implémentations peuvent contenir des données, des threads et des groupes de thread.

```
process Procedure
end Procedure ;
process implementation Procedure . Impl
      subcomponents
      Capteur_A : thread Capteur ;
      Donnees : thread Donnees ;
      Alrm : thread Alrm_Thread ;
      connections
      data port Capteur_A. outp−>Donnees . inpA ;
      event port Sensor_A. launch−>Alrm. launch A;
end Pa r t i t i on . Impl ;
```

3.1.2 Composants de la plateforme d'exécution

Les composants de la plate-forme d'exécution AADL sont : les processeurs, les devices, les mémoires et le bus, et leur représentation graphique est visible dans la Figure 2.3 qui illustre la représentation des différentes catégories de composants de plateforme selon :

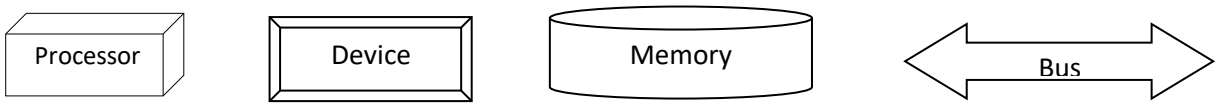

Figure 2.3 Représentation graphique des composants matériels

3.1.2.1 **Processor :** Bien que spécifié comme composant matériel, le processeur contient également le logiciel responsable de la planification et de l'exécution des threads.

Il peut avoir des ports et des fonctionnalités d'accès au bus, et sa mise en œuvre peut contenir des sous-composants de mémoire.

3.1.2.2 **Device :** Représente une interface avec l'environnement, allant d'un simple capteur aux abstractions de sous-systèmes entiers. Ses fonctionnalités peuvent être des ports, des accès aux bus et des sous-programmes serveur. Un appareil ne peut contenir aucun sous-composant. Il permet de représenter un élément dont la structure interne est ignorée.

3.1.2.3 **Memory :** Représente une unité de stockage générique. Son seul type de fonctionnalité autorisé est l'accès au bus, donc une mémoire communique avec un processeur soit si elle est contenue dans une implémentation de processeur, soit si la mémoire et le processeur ont accès à un bus commun. Une implémentation de mémoire peut contenir d'autres composants de mémoire.

3.1.2.4 **Bus** : Abstraction du matériel et des logiciels responsables de la communication entre les autres composants de la plateforme d'exécution. Un bus AADL n'a pas de fonctionnalités (sauf les accès au bus lorsqu'il a besoin d'accéder à un autre bus), et son implémentation ne peut pas non plus avoir de sous-composants, donc ses liens avec les autres composants se font tous avec des associations de propriétés.

3.1.2.5 **System** : Les systèmes AADL permettent de rassembler différents composant pour former des blocs logiques d'entités ; ils facilitent la structuration de l'architecture. Contrairement aux autres catégories de composants, la sémantique des systèmes n'est donc pas associée à des éléments précis. Le composant système représente un composant composite comme un ensemble de composants logiciels et de plate-forme d'exécution ou de composants système.

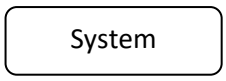

Figure 2.4 : représentation graphique du système

4. Types de composants et implémentation

Un type de composant spécifie une interface fonctionnelle en termes de fonctionnalités alors qu'une implémentation de composant spécifie une structure interne qui fait référence à un type défini ailleurs (liste 3.1). Elle représente une spécification de plusieurs implémentations d'un type de composant qui peuvent être déclarées, permettant de modéliser plusieurs variantes avec les mêmes interfaces externes car chaque implémentation fournit une réalisation d'un composant qui satisfait la même interface spécifiée par le type de composant.

De plus, une implémentation de composant peut étendre et affiner d'autres implémentations de composants précédemment déclarées. Les implémentations étendues (déclarées avec la sous-clause extends) héritent des caractéristiques de l'implémentation du composant d'origine et de tous ses prédécesseurs. On peut résumer la syntaxe générale des déclarations du package, du type et de l'implémentation des composants comme suit

• Déclaration d'un package nom_du_package:

```
package nom_du_package
 public …
end nom_du_package;
```

• Déclaration d'un type d'un composant nom_du_composant (ex :process)

```
process nom_du_composant
end nom_du_composant;
```

• Déclaration d'une implémentation de composant nom_du_composant.impl_nom, avec un sous composant subcompo_nom.

```
process implementation nom_du_composant.impl_nom
subcomponents subcompo_nom: thread other_cpt_id;
                    -- ou other_cpt_id.other_impl_id
 end nom_du_composant.impl_nom;
```

(Les commentaires commencent par **- -**).

Exemple

```
processor i486
end i486 ;
processor i486dx extends i486
end i486dx ;
processor implementation i486 . I n t e l
end i486 . I n t e l ;
processor implementation i486 .AMD
end i486 .AMD;
processor implementation  i486dx . I n t e l extends i486 . I
n t e l
end i486dx . I n t e l :
```

5. Structure interne des composants

Les composants peuvent être décomposés hiérarchiquement en collections ou assemblages de sous-composants en interaction. Un sous-composant déclare un composant contenu dans un autre composant. Un sous-composant nomme un type de composant et une implémentation de composant pour spécifier une interface et une implémentation pour le sous-composant. Ainsi, les types de composants et les implémentations agissent comme des classificateurs de

composants. La hiérarchie d'une instance de système est basée sur l'ensemble des sous-composants de l'implémentation du système de niveau supérieur. Il est complété en parcourant itérativement l'arborescence des classificateurs de composants spécifiés en commençant par les sous-composants de mise en œuvre du système de niveau supérieur. Les sous-composants permettent de spécifier les systèmes sous forme de hiérarchie de confinement statique et arborescente. L'AADL permet également aux composants de référencer des sous-composants qui ne sont pas contenus exclusivement dans le composant. Cela permet d'accéder ou d'utiliser un composant dans plusieurs composants.

Par exemple, les éléments de données statiques contenus dans le texte source et représentés dans AADL en tant que composants de données peuvent être utilisés par des threads dans différents processus (dont les espaces d'adressage protégés peuvent autrement être distincts).

```
process processus_a
end processus_a ;
thread thread_a
end thread_a ;
process implementation processus_a . impl
subcomponents
thread1 : thread thread_a . impl ;
end processus_a . impl ;
thread implementation thread_a . impl
c a l l s
sequence : { appel1 : subprogram sp_a ;
appel2 : subprogram sp_a ; } ;
end thread_a . impl ;
subprogram sp_a
end sp_a ;
```

6. *Les interfaces(features*)

Ce standard définit plusieurs catégories d'interfaces : port de données, port d'événement, port de données et d'événement, groupe de ports, paramètre de sous-programme et accès aux données fourni et requis.

6.1 ***Les ports de données :*** représentent des points de connexion pour le transfert de données d'état telles que les données d'un capteur.

6.2 ***Les ports d'événements*** : représentent des points de connexion pour le transfert de contrôle via des événements déclenchés qui peuvent déclencher l'envoi de threads ou une transition de mode.

6.3 ***Les ports de données d'événements :*** représentent des points de connexion pour le transfert d'événements avec des données, c'est-à-dire des messages qui peuvent être mis en file d'attente.

6.4 ***Les groupes de ports :*** prennent en charge le regroupement de ports, de sorte qu'ils puissent être connectés à d'autres composants via une seule connexion.

6.5 **Les sous-programmes de données :** représentent des points d'entrée vers des séquences de code dans le texte source associées à un type de données.

6.6 ***Les sous-programmes*** représentent des points de connexion pour les appels/retours synchrones entre les threads ; dans certains cas, l'appel/le retour peut être effectué à distance.

6.7 ***Les paramètres de sous-programme*** : représentent les paramètres d'entrée et de sortie d'un sous-programme.

6.8 ***L'accès aux composants de données :*** représente l'accès fourni et requis aux données partagées.

6.9 ***L'accès aux composants de bus*** : représente l'accès fourni et requis aux bus pour les processeurs, la mémoire et les périphériques.

6.10 ***L'accès aux composants de sous-programme*** : représente la liaison d'un appel de sous-programme au sous-programme appelé. (Liste 3.3 et Figure 3.5)

Listing 3.3 – Examples of the interface

```
system systeme_a
features
a : in data por t ;
end systeme_a ;
system systeme_b extends systeme_a
features
a : refined to in data por t donnee ;
b : requires data access donnee ;
c : port group groupe_de_ports ;
end systeme_b ;
data donnee
end donnee ;
port group groupe_de_ports
features
p1 : in data por t donnee ;
p2 : in out event data port donnee ;
end groupe_de_ports ;
process proces s_a
features
e : in event port ;
s : out event data port ;
end proces s_a ;
subprogram sous_programme_a
end sous_programme_a ;
thread thread_a
features
sp : server subprogram
sous_programme_a ;
end thread_a ;
```

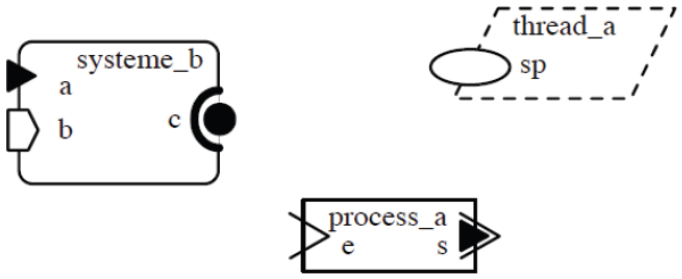

Figure 2.5 : Connexion entre composants.

7. Connections

Les connexions AADL spécifient des modèles de contrôle et de flux de données entre les composants individuels au moment de l'exécution. Les connexions permettent de relier les interfaces des différents souscomposants à celles d'autres sous-composants ou aux interfaces du composant parent. Une connexion est orientée et doit éventuellement être nommée.

Une connexion sémantique peut être établie entre deux threads, entre un thread et un périphérique ou un processeur.

Les connexions de ports : entre un composant de données et les threads qui accèdent au composant de données.

Les connexions d'accès aux données : entre un composant de bus et des bus, de la mémoire, un processeur et un composant de périphérique.

Les connexions d'accès au bus : entre un composant de sous-programme et des threads qui nécessitent un accès par appel au sous-programme, ou entre le port d'événement d'un thread, d'un périphérique ou d'un processeur et une transition de mode pour les connexions de transition de mode.

Une connexion sémantique est représentée par un ensemble d'une ou plusieurs déclarations de connexion qui suivent la hiérarchie des composants depuis la source de connexion ultime jusqu'à la destination de connexion ultime. Par exemple, dans la figure 2.6, il y a une déclaration de connexion depuis un port de sortie de thread dans Thread P vers un port de sortie de processus contenant dans Process A. Cette connexion se poursuit avec une déclaration de connexion dans System Simple depuis le port de sortie de Process A vers le port d'entrée de Process B. La déclaration de connexion se poursuit dans Process B jusqu'au thread dans le port contenu dans Thread Q. Collectivement, cette séquence de connexions définit une seule connexion sémantique entre Thread P et Thread B. Les threads, processus, systèmes et ports sont affichés en notation graphique AADL.

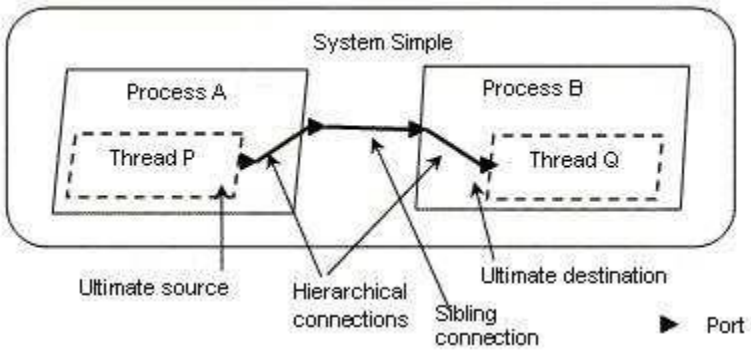

Figure 2.6 Les connexions de ports

8. *Le flux*

Les spécifications de flux décrivent un flux d'informations observable de l'extérieur en termes de logique d'application via un composant. De tels flux logiques peuvent être réalisés via des ports et des connexions de différents types de données et une combinaison de ports de données, d'événements et de données d'événement, ainsi que des flux via des composants de données partagés. Les spécifications de flux représentent les sources de flux, c'est-à-dire les flux provenant de l'intérieur d'un composant, les puits de flux, c'est-à-dire les flux se terminant dans un composant, et les chemins de flux, c'est-à-dire les flux traversant un composant depuis ses ports entrants vers ses ports sortants. Les flux décrivent des séquences de flux réelles à travers des composants et des ensembles de composants sur une ou plusieurs connexions. Ils sont déclarés dans les implémentations des composants. Les séquences de flux prennent deux formes : implémentation de flux et flux de bout en bout. Une implémentation de flux décrit comment une spécification de flux d'un composant est réalisée dans son implémentation de composant.

Un flux de bout en bout spécifie un flux qui commence dans un sous-composant et se termine dans un autre sous-composant. Les spécifications de flux, les implémentations de flux et les flux de bout en bout peuvent avoir des valeurs attendues et réelles pour les propriétés liées au flux, par exemple la latence ou l'accumulation d'erreurs d'arrondi.

9. *Outils disponibles*

Depuis le début des efforts de normalisation, certains outils ont été développés pour fonctionner avec AADL. L'outil principal au moment de ce travail est l'Open-Source AADL Tool Environment (OSATE, disponible sur (SAE 2004)) qui consiste en une série de plug-ins fonctionnant sous la plateforme Eclipse. OSATE permet de créer des spécifications textuelles AADL (fichiers .aadl) et des modèles d'objets liés à XML (fichiers .aaxl). Il comprend un analyseur pour vérifier la syntaxe et certains aspects sémantiques de la spécification, et est capable de traduire un fichier .aadl en un fichier .aaxl et vice-versa. OSATE comprend également des plug-ins de prototypes d'analyse qui permettent des vérifications de modèle de base, telles que la liaison entre les threads et les processeurs, les contrôles de flux et l'allocation de ressources. Tous les plug-ins d'analyse sont décrits dans les fichiers d'aide d'OSATE.

Un éditeur graphique AADL est fourni par le projet TOPCASED. Cet éditeur graphique fonctionne avec OSATE, les diagrammes peuvent donc être basés sur des spécifications AADL réalisées avec OSATE et des spécifications textuelles peuvent également être créées à partir de diagrammes AADL.

D'autres outils méritent d'être mentionnés : l'outil d'analyse d'ordonnancement Cheddar (Singhoff et al.), développé par l'Université de Brest et qui est désormais également capable de faire des analyses d'ordonnancement dans les spécifications AADL ; et l'Architecture Description Simulation (ADeS) (Axlog), développée par Axlog. ADeS est un plugin Eclipse qui fonctionne avec OSATE et simule l'exécution d'une spécification AADL.

10.Conclusion

Nous avons introduit le langage AADL. Ce langage utilise les fonctionnalités classiques d'un langage de description de l'architecture. Ce langage suggère la notion de motifs pour décrire un ensemble de configurations dynamiques, bien que le système soit globalement statique.
De plus, AADL est un langage de description et d'analyse des systèmes temps réel critiques. La norme définit le modèle d'exécution sous-jacent au langage et définit le comportement des modèles AADL. Enfin, il fournit des mécanismes d'extension pour enrichir à des fins dédiées