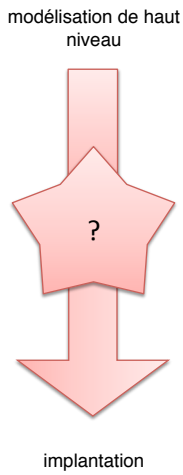
AADL: Architecture Analysis & Design Language

Bechir Zalila, bechir.zalila@enst.fr
Laurent Pautet, laurent.pautet@enst.fr
Thomas Vergnaud, thomas.vergnaud@enst.fr

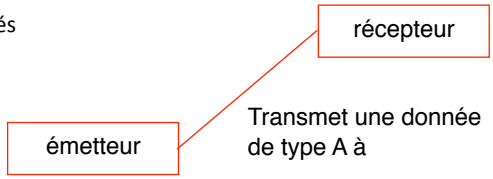
Problématique

- Approches formelles de description
  - SCADE, SDL, réseaux de Petri, etc.
  - couplés à des générateurs de code
  - mise en place de la répartition
- Eléments concrets de mise en œuvre
  - réseaux, intergiciels, etc.
  - contraintes sur l'application répartie
  - adaptation aux besoins de l'application
- Comment rassembler ces deux aspects?
  - passer de la description formelle à l'implantation
  - évaluer les performances de l'application
    - temps d'exécution
    - place en mémoire
    - fiabilité



Les langages de description d'architecture (ADL) - 1

- La conception des systèmes devient complexe
  - difficulté de compréhension du système
  - preuve de cohérence de l'architecture du système
- Un formalisme de description est nécessaire
  - pour décrire l'architecture
  - pour analyser et vérifier ses propriétés
- 3 éléments principaux :
  - les composants
  - les liens entre les composants
  - une sémantique associée à ces composants



Les langages de description d'architecture (ADL) - 2

- ADL formels
  - Formalisent la description du fonctionnement d'un système
  - S'intègrent mal dans une démarche de génération automatique
  - Wright, Rapide
- ADL concrets
  - Décrivent l'architecture afin de la générer automatiquement
  - UML 2, ***AADL***
- ADL restreints
  - Décrivent l'assemblage de composants logiciels ou la cohérence de l'application
  - ArchJava, Fractal

Architecture Analysis & Design Language

- Anciennement « Avionics Architecture Description Language »
- Evolution de MetaH, qui était développé par Honeywell
  - Dédié aux systèmes répartis embarqués temps-réel
  - Décrit des éléments matériels et logiciels
  - Conçu pour permettre la génération de systèmes exécutables
    - expression des caractéristiques des éléments du système
    - traduction en langage de programmation
- Plusieurs représentations
  - representation textuelle
    - pour contrôler tous les détails du système
  - représentation en XML
    - pour l'interopérabilité entre outils
  - représentation graphique
    - convenable pour avoir une vision globale du système
  - profil UML 2, et représentation en UML 1.4
- Version 1.0 publiée en 2004
- Version 2 sera publiée en 2009

AADL 1.0 (http://www.aadl.info)

- Publié en 2004
- Déjà utilisé par de grands projets
  - COTRE (Airbus)
  - ASSERT (ESA, EADS, ENST, INRIA, etc.)
  - Topcased (Airbus, CNES, ENST, etc…)
  - …
- Quelques outils disponibles
  - OSATE : outil de référence pour Eclipse (SEI/CMU)
    - syntaxe textuelle et graphique
    - vérifications syntaxiques et sémantiques générales
    - plusieurs extensions pour la vérification d'architectures
  - STOOD : outil de modélisation basé sur la méthode HOOD (Ellidiss)
    - outil graphique (syntaxe UML & HOOD)
    - générateur de code pour des applications monolithiques
  - Ocarina : suite d'outils pour générer des applications (ENST)
    - noyau central pouvant être intégré dans différentes applications
    - plusieurs générateurs d'applications (Ada et C)

Principes généraux

- Composants reliés entre eux
  - Connexions reliant les interfaces (ports, paramètres…).
  - Chaque composant, défini par un type + une ou plusieurs implémentations
- Les composants pouvant contenir des sous-composants
- Possibilité d'attribuer des propriétés à chaque élément
  - composant, sous-composant, connexion, port…
- Caractéristiques de l'implémentation pouvant varier
  - selon des modes de configuration
  - selon les propriétés

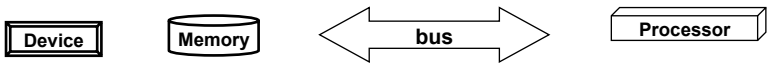
- But: modéliser la réalité
  - générer facilement un système fonctionnel à partir de sa description.
  - Tous les éléments correspondent à quelque chose de concret
- AADL: un langage descriptif
  - les éléments peuvent être donnés dans n'importe quel ordre

Catégories de composants

- Éléments de base d'une description architecturale
- Déclaration en plusieurs parties
  - component type : interface
  - component implementation : structure interne
  - un type peut avoir plusieurs implémentations
- Plusieurs catégories
  - matériels (*execution platform components*)
    - processor, memory, bus, device
  - logiciels (*software components*)
    - thread, data, process, thread group, subprogram
  - systèmes (*system composition*)
    - system

Les composants matériels

- Plusieurs catégories
  - processor : micro-processeur + ordonnanceur
  - memory : disque dur, mémoire vive, etc.
  - bus : réseau, etc.
  - device : composant dont on ignore la structure interne
- Un processor modélise processeur + noyau contenant entre-autres un ordonnanceur.
- Un device sert typiquement à modéliser un capteur

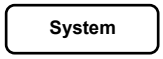


Les composants logiciels

- Plusieurs catégories
  - thread : fil d'exécution (ou thread dans les noyaux)
  - data : structure de données
  - process : processus, un espace mémoire pour l'exécution des threads qu'il contient
  - thread group : crée une hiérarchie dans les threads
  - subprogram : procédure, comme pour les langages de programmation. N'as pas de valeur de retour
- Un process doit contenir au moins un thread.

Les systèmes

- Permettent de structurer la description.
- Contiennent les composants qui peuvent être manipulés de façon indépendante :
  - ⌘ system
  - ⌘ processor, memory, device, bus
  - ⌘ process, data
- MAIS PAS :
  - ⌘ thread
  - ⌘ thread group
  - ⌘ subprogram



Les sous-composants

- Un composant peut avoir des sous-composants
  - ⌘ décrits dans les implémentations des composants
- Une modélisation AADL est une arborescence de composants

data	data
thread	data
thread group	data, thread, thread group
process	thread, thread group
processor	memory
memory	memory
system	tous sauf subprogram, thread et thread group

Exemple de composants

```
thread execution_thread
end execution_thread;

process a_process
end a_process;

process implementation a_process.one_thread
subcomponents
  thread1 : thread execution_thread;
end a_process.one_thread;

process implementation a_process.two_threads
subcomponents
  thread1 : thread execution_thread;
  thread2 : thread execution_thread;
end a_process.two_threads;
```


Appels de sous-programmes

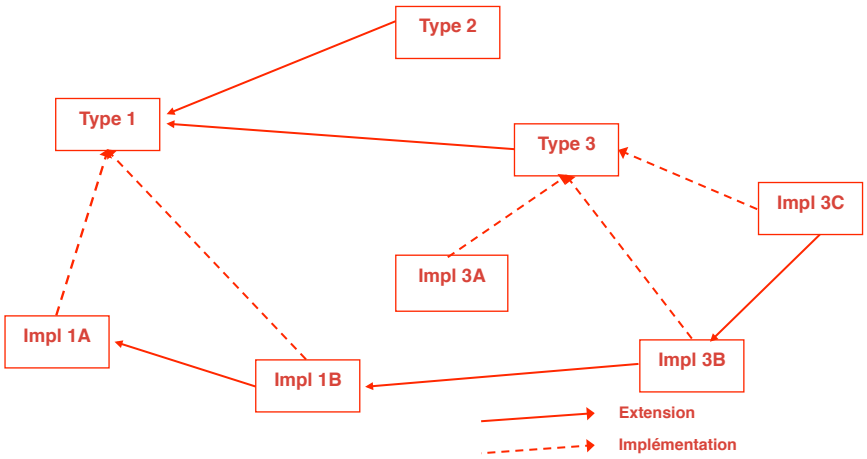
- Une implantation de sous-programme ou de thread peut contenir des séquences d'appels à des sous-programmes
- L'ordre des appels est important
- Description du flux d'exécution dans les composants

```
subprogram spg1 end spg1;
subprogram spg2 end spg2;
thread a_thread end a_thread;

thread implementation a_thread.example
calls {
  call1 : subprogram spg1;
  call2 : subprogram spg2;
  call3 : subprogram spg1;
};
end a_thread.example;
```

Extension des composants

- un composant AADL peut étendre un autre (*component extension*)



Exemple d'extension de composant

```
thread execution_thread
end execution_thread;

process a_process
end a_process;

process implementation a_process.one_thread
subcomponents
  thread1 : thread execution_thread;
end a_processus.one_thread;

process implementation a_process.two_threads
extends a_process.one_thread
subcomponents
  thread2 : thread execution_thread;
end a_process.two_threads;
```

- `a_process.two_threads` contient **deux** threads

Les features— les ports

- Les ports modélisent les échanges d'information.
  - ⌘ data : transport de données ; comme dans un circuit électronique
  - ⌘ event : émission d'un signal
  - ⌘ event data : signal + données ; comparable à un message
- les ports peuvent être déclarés en
  - ⌘ entrée (`in`)
  - ⌘ sortie (`out`)
  - ⌘ entrée-sortie (`in out`)
- Contrairement aux data ports, les event ports peuvent être mis en file d'attente
  - ⌘ L'information envoyée sur un port data peut écraser la précédente
  - ⌘ La longueur de la file peut être spécifié par l'intermédiaire d'une *propriété* AADL
  - ⌘ Si la file d'attente est pleine, une des politique suivantes peut être adoptée:
    - ○ DropOldest: On supprime le message le plus ancien
    - ○ DropNewest: On supprime le message le plus récent

Features: paramètres & sous-programmes d'interface

- Pour des sous-programmes, on parle de paramètres (**`parameters`**)
- Un paramètre s'utilise comme un port de donnée
  - ⌘ data port
  - ⌘ event data port
- Les paramètres peuvent être `in`, `out` ou `in out`
- Un composant thread ou data peut offrir des sous-programmes comme interface
  - ⌘ un thread serveur
    - ○ p.ex. dans le cas d'un appel de procédure distante (RPC)
  - ⌘ un composant de donnée proposant des méthodes d'accès
    - ○ analogie avec les classes des langages objets

Features: les accès aux composants

- Un composant peut indiquer qu'il requiert (`requires`) ou qu'il fournit (`provides`) un accès à un sous-composant
  - un bus, p.ex. pour un processor ou une memory
  - une data, p.ex. pour une donnée partagée entre plusieurs threads
- On exprime ainsi l'obligation de brancher un composant avec un autre pour obtenir un système cohérent
  - un processor, une memory ou un device doivent être connectés aux autres composants matériels par l'intermédiaire d'un bus

Features: les groupes de ports

- Regroupement de ports associés entre eux
  - facilite la manipulation au niveau de la description
  - analogie avec un câble

Exemple de features

```
data pressure
end pressure;

data altitude
end altitude;

thread altimeter
features
  P : in event data port pressure;
  A : out data port altitude;
end altimeter;
```

```
thread client_altitude
features
  A : in data port altitude;
end client_altitude;

device pressure_sensor
features
  P : out event data port pressure;
end pressure_sensor;

subprogram compute_altitude
features
  P : in parameter pressure;
  A : out parameter altitude;
end compute_altitude;
```


Les connexions

- Pour relier les « features » entre elles
  - ⌘ ports
  - ⌘ paramètres
  - ⌘ sous-programmes d'interface
  - ⌘ accès aux sous-composants
  - ⌘ groupes de ports
- Les connexions ont une direction
  - ⌘ les entrées sont reliées aux sorties de sous-composants
  - ⌘ les entrées d'interfaces sont reliées aux entrées des sous-composants ; idem pour les sorties
- Les features de sortie peuvent être « 1 vers n »
- `event data ports` & `event ports` entrants
  - ⌘ « n vers 1 » car gestion de files d'attente
- les autres features entrantes
  - ⌘ « 1 vers 1 »

Exemple de connexion (1)

```
process manager
features
  P : in event data port pressure;
end manager;

thread implementation altimeter.basic
calls {
  appli : subprogram compute_altitude;
};
connections
  parameter P -> appli.P;
  parameter appli.A -> A;
end altimeter.basic;

process implementation manager.altitude
subcomponents
  client1 : thread client_altitude;
  server1 : thread altimeter.basic;
connections
  pressure_input : event data port P -> server1.P;
  data port server1.A -> client1.A;
end manager.altitude;
```


Exemple de connexion (2a)

```
data signal
end signal;

port group signal_DB9
features
  CD  : in  data port signal;   -- carrier detection
  RD  : in  data port signal;   -- data reception
  TD  : out data port signal;   -- data transmission
  DTR : out data port signal;   -- ready to transmit data
  DSR : in  data port signal;   -- ready to send data
  RTS : out data port signal;   -- transmision request
  CTS : in  data port signal;   -- ready for transmission
  RI  : in  data port signal;   -- reception indicator
end signal_DB9;

port group signal_DB9_inverse inverse of signal_DB9
end signal_DB9_inverse;
```

Exemple de connexion (2b)

```
system serial_card
features
  plug : port group signal_DB9;
end serial_card;

system serial_wire
features
  plug : port group signal_DB9_inverse;
end serial_wire;

system global
end global;

system implementation global.basic
subcomponents
  card : system serial_card;
  wire : system serial_wire;
connections
  port group card.plug -> wire.plug;
end global.basic;
```


Exemple de connexion (3)

```
data a_data
end a_data;

subprogram prog1
features
  input : in parameter a_data;
  output : out parameter a_data;
end prog1;

subprogram prog2
features
  input : in parameter a_data;
  output : out parameter a_data;
end prog2;
```

```
subprogram implementation prog2.impl
calls {
  a_call : subprogram prog1;
};
connections
  parameter input -> a_call.input;
  parameter a_call.output -> output;
end prog2.impl;
```

Les modes

- Les modes permettent de modéliser la reconfiguration du système en fonction d'événements
  - définis au niveau des composants
  - seuls les événements (event ports) peuvent déclencher un changement de mode
  - certains sous-composants, connexions, etc. ne sont activés que dans certains modes
- Les changements de modes permettent de représenter des architectures dynamiques
  - évolution dans la configuration de l'architecture
  - ensemble déterminé de configurations possibles

Exemple de modes

```
thread execution_thread
end execution_thread;

process a_process
features
  multi_thread : in event port;
  mono_thread : in event port;
end a_process;

process implementation a_process.configurable
subcomponents
  thread_1 : thread execution_thread;
  thread_2 : thread execution_thread in modes (multitask);
modes
  monotask : initial mode;
  multitask : mode;
  monotask -[ multi_thread ]-> multitask;
  multitask -[ mono_thread ]-> monotask;
end a_process.configurable;
```

Les flots (1)

- Les flots permettent de matérialiser les chemins à travers les éléments d'une description. Ils suivent plus ou moins les connexions.
  - ne modélisent rien de concret
  - Une forme de redondance pour pouvoir analyser plus facilement les flots de données et y associer des propriétés
- on peut décrire :
  - le début d'un flot (`flow source`)
  - le milieu d'un flot (`flow path`)
  - la fin d'un flot (`flow sink`)
  - un flot complet (`end to end flow`)

Les flots (2)

- Les *component types* contiennent
  - des flow specifications (source, sink et path) qui ne font intervenir que des `features`
- les *component implementations* contiennent
  - des flow implementations (source, sink et path) qui font intervenir des features, des connections et des flots de sous-composants
  - des end to end flows qui commencent par un flow source et finissent par un flow sink.

Exemple de flow specification

```
process foo
features
  InitCmd : in  event      port;
  Signal  : in        data port signal_data;
  Result1 : out       data port position.radial;
  Result2 : out       data port position.cartesian;
  Status  : out event      port;
flows
  Flow1 : flow path   Signal -> Result1;
  Flow2 : flow path   Signal -> Result2;
  Flow3 : flow sink   InitCmd;
  Flow4 : flow source Status;
end foo;
```


Exemple de flow implementation

```
process implementation foo.basic
subcomponents
A: thread bar.basic;
-- bar has a flow path fs1 from port p1 to p2
-- bar has a flow source fs2 to p3
C: thread baz.basic;
B: thread baz.basic;
-- baz has a flow path fs1 from port p1 to p2
-- baz has a flow sink fsink in port reset
connections
Conn1        : data  port signal  -> A.p1;
Conn2        : data  port A.p2    -> B.p1;
Conn3        : data  port B.p2    -> result1;
Conn4        : data  port A.p2    -> C.p1;
Conn5        : data  port C.p2    -> result2;
Conn6        : data  port A.p3    -> status;
connToThread : event port initcmd -> C.reset;
flows
Flow1: flow path signal -> conn1 -> A.fs1 -> conn2 -> B.fs1 -> conn3 -> result2;
Flow2: flow path signal -> conn1 -> A.fs1 -> conn4 -> C.fs1 -> conn5 -> result2;
Flow3: flow sink initcmd -> connToThread -> C.fsink;
-- a flow source may start in a subcomponent,
-- i.e., the first named element is a flow source
Flow4: flow source A.fs2 -> connect6 -> status;
end foo.basic;
```

Les propriétés en AADL (1)

- Les propriétés peuvent être associées à quasiment tous les éléments d'une description
- Une propriété permet d'associer une valeur d'un certain type (ou non typée) à un nom.
  - la norme prévoit un ensemble de propriétés standard
  - il est possible de définir de nouvelles propriétés dans des ensembles de propriétés (property sets)
- Une propriété peut ne s'appliquer qu'à un ensemble de catégories d'éléments (p.ex. les processeurs)

Les propriétés en AADL (2)

- Le type d'une propriété peut être
  - un booléen : `aadlboolean`
  - un entier : `aadlinteger`
  - un réel : `aadlreal`
  - une chaîne de caractères : `aadlstring`
  - une énumération : `enumeration`
  - une catégorie d'élément : `classifier` (composant, connexion, etc.)
  - une référence à un element: `reference` (composant…)
  - une plage de valeurs : `list of` …

Exemples de propriétés prédéfinies

```
Supported_Queue_Processing_Protocols :
  type enumeration (FIFO, <project_related>);

Queue_Processing_Protocol :
  Supported_Queue_Processing_Protocols => FIFO
    applies to (event port, event data port, subprogram);

Source_Text : inherit list of aadlstring
  applies to (data, port, subprogram, thread, thread group,
              process, system, memory, bus, device, processor,
              parameter, port group);

Max_Thread_Limit : constant aadlinteger => <project_related>;

Thread_Limit : aadlinteger 0 .. value (Max_Thread_Limit)
  => value (Max_Thread_Limit) applies to (processor);
```


Les propriétés en AADL (3)

- Pouvant dépendre d'un mode
- Pouvant se propager dans les sous-composants
  - mot-clef **`inherit`**
- Pouvant s'ajouter ou remplacer la valeur définie dans un composant père
  - extension de composant
- Pouvant être déclarées
  - dans un composant,
  - au niveau de la déclaration d'un sous-composant, une connexion
  - dans un composant père et s'appliquer à un de ses sous-élément (sous-composant, feature, appel à un sous-programme…)
- les propriétés permettent (entre autres) d'indiquer le déploiement des composants logiciels sur les composants matériels

Exemples d'associations de propriétés

```
processor a_processor
end a_processor;

processor implementation a_processor.simple
properties
  Thread_Swap_Execution_Time => 0ms .. 10 ms;
end a_processor.simple;

process a_process
end a_process;

system global
end global;

system implementation global.simple
subcomponents
  processor1 : processor a_processor.simple {Thread_Limit => 3;};
  process1   : process a_process;
properties
  Actual_Processor_Binding => reference processor1 applies to process1;
end a_process.simple;
```


Exemple d'ensemble de propriétés

```
property set our_properties is
  pressure     : type aadlinteger units (Pa, HPa => 100 * Pa);
  --  Définition des unités Pascal et Hecto-Pascal

  pressure_max : pressure applies to (device);
end our_properties;

system a_system
end a_system;

device a_sensor
end a_sensor;

system implementation a_system.with_a_sensor
subcomponents
  the_sensor : device a_sensor
    {our_properties::pressure_max => 1020 hPa;};
end a_system.with_a_sensor;
```

Les annexes

- Permettent d'enrichir la description en utilisant une syntaxe autre que celle d'AADL
  - ⌘ p.ex. Z, OCL, etc.
- Peuvent être contenues dans les composants et les groupes de ports

```
thread Collect_Samples
features
  Input_Sample   : in  data port Sample;
  Output_Average : out data port Sample;
annex OCL {**
  pre:  0 < Input_Sample < maxValue;
  post: 0 < Output_Sample < maxValue;
**};
end Collect_Samples;
```


Annexes et propriétés

- Deux façons d'enrichir une description AADL
  - ⌘ utiliser des annexes
  - ⌘ utiliser des propriétés
- Une annexe n'est pas obligatoirement interprétée par les outils d'exploitation
  - ⌘ mais sa détection par un outils qui ne peut pas l'interpréter NE DOIT PAS causer des erreurs de syntaxe.
- On peut associer des propriétés à quasiment tous les éléments d'une description
- Une annexe sert à ajouter des précisions facultatives. Une propriété est généralement très liée à la description architecturale
- Les annexes et propriétés permettent une grande souplesse
  - ⌘ attention à ne pas détourner leur utilisation
    - ○ décrire tout le comportement d'un sous-programme avec des annexes au lieu d'utiliser les séquences d'appel
    - ○ utiliser une propriété pour référencer un fichier qui contient la description en langage naturel de l'architecture d'un composant

Organisation d'une description

- Les paquetages (packages)
  - matérialisent des espaces de nom
  - une partie publique : visible de partout
  - une partie privée : uniquement visible depuis le paquetage
    - composants
    - groupes de ports
    - Propriétés
- Espace de nom anonyme (anonymous namespace)
  - le plus haut niveau de la description
    - paquetages
    - composants
    - groupes de ports
    - ensembles de propriétés
- Les systèmes permettent de structurer l'architecture
- Les paquetages permettent de structurer la description

Exemple d'utilisation des paquetages

```
package machines
public
  system a_machine
  end a_machine;

  system implementation a_machine.mono_processor
  subcomponents
    the_processor : processor machines::elements::a_processor.specific;
  end a_machine.mono_processor;
private
  system a_private_machine
  end a_private_machine;
end machines;

package machines::elements
public
  processor a_processor
  end a_processor;

  processor implementation a_processor.specific
  end a_processor.specific;
end machines::elements;
```

Description complète

- Une modélisation AADL est un ensemble de déclarations de composants
- Les sous-composants sont des instances des déclarations
- Un système doit jouer le rôle de racine pour l'architecture
  - pas d'interface
  - une implémentation contenant les sous-composants

```
system global_system
end global_system;

system implementation global_system.two_machines
Subcomponents
  machine_1 : system machines::a_machine.mono_processor;
  machine_2 : system machines::a_machine;
end global_system.two_machines;
```


Synthèse des sous-éléments (1)

- component types
  - features
  - flows
  - properties
  - annexes
- port groups
  - features (et/ou inverse of)
  - properties
  - annexes
- component implementations
  - subcomponents
  - calls
  - connections
  - flows
  - modes
  - properties
  - annexes

Synthèse des sous-éléments (2)

- properties : tous les composants
- modes : tous les composants
- flows : les composants d'exécution
  - ⌘ subprogram, thread, thread group, process, processor, device, system
- connections : tous les composants qui ont des sous-composants
- subprogram calls : thread, subprogram
- features : propres à chaque type de composant

Instanciation de l'architecture

- Une description AADL est une suite de déclarations
  - ⌘ analogie avec un diagramme de classes UML
  - ⌘ analyse et manipulation limitées
    - ○ certaines propriétés sont associées à un sous-composant particulier
- Il est nécessaire d'instancier la modélisation
  - ⌘ arborescence d'instances de composants correspondants aux déclarations de sous-composants
    - ○ Valeurs de propriétés résolues
  - ⌘ les entités qui ne sont pas instanciées:
    - ○ les composants data associés aux features car il représentent un type.
  - ⌘ on manipule alors l'architecture telle qu'elle doit être dans la réalité

Génération d'un système exécutable

- Une modélisation AADL décrit
  - les caractéristiques des composants
  - les connexions
- Les propriétés standard permettent d'associer un code source à chaque composant
  - VHDL, …
  - Ada, C, …
- Ces codes sources indiqués en propriétés doivent se conformer aux spécifications AADL (signatures des sous-programmes…)
  - écrits à la main
  - générés automatiquement par d'autres outils
- Une annexe du standard donne les correspondances entre les syntaxes AADL et Ada ou C
- On peut indiquer les codes sources des composants et de générer une application pour un exécutif AADL

Paramètres pour la génération

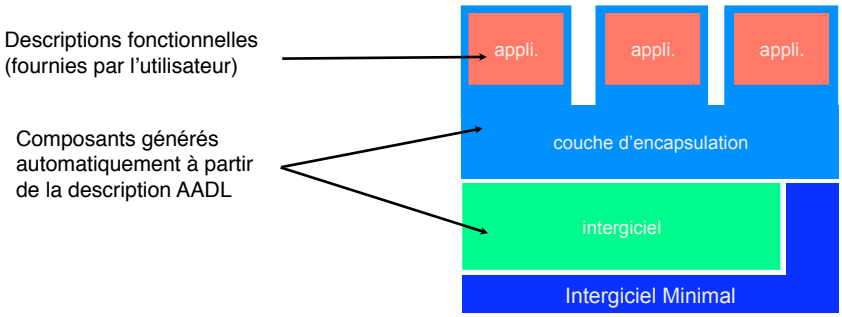
- Composants
  - matériels : fournissent les informations de déploiement
    - caractéristiques des machines
    - connexions sur les réseaux
    - …
  - logiciels
    - correspondent aux applications dont il faut générer le code
    - processus : modélisent les nœuds (partitions)
    - threads : éléments actifs
    - sous-programmes : éléments réactifs des applications
  - systèmes
    - éléments de structure non fonctionnels de l'architecture
- Interfaces et connexions entre les processus
  - correspondent aux modèles de répartition utilisés

Utilisation d'un intergiciel

- Exécutif AADL = intergiciel
  - thread AADL fourni par l'intergiciel ou le noyau
  - communications assurées par l'intergiciel
- Générateur de code:
  - génération de code applicatif à partir des données et des sous-programmes AADL
  - génération de code intergiciel pour l'adapter à l'application AADL
    - interfaçage des threads et des processus AADL
    - traduction des mécanismes de communication en fonction de l'intergiciel
- Les composants applicatifs sont pilotés par l'exécutif
  - code utilisateur encapsulé, n'a aucun contrôle sur l'exécutif
  - possibilité de vérification
    - à condition de connaître les correspondances entre un thread AADL et la configuration correspondante de l'intergiciel
  - le contrôle par l'intergiciel limite la liberté de modélisation
    - certaines constructions architecturales ne sont pas possibles

Exécutif pour les nœuds applicatifs

- L'exécutif AADL doit assurer deux tâches
  - contrôle de l'exécution des threads (ordonnancement)
  - prise en charge des communications (inter- et intra- nœuds)



Vérification de la description AADL

- Vérification syntaxique
  - valeur des propriétés bornées
  - ensemble de propriétés AADL_Project caractéristique des dimensions de l'exécutif
    - protocoles possibles, etc.
- Vérification de la cohérence des spécifications
  - tailles mémoire des données et des mémoires
  - temps d'exécution des sous-programmes et des threads
  - nombre de threads par processeur
- Vérification des limitations relatives à l'exécutif
  - bonne utilisation des propriétés vis-à-vis des capacités de l'exécutif
  - bonne utilisation des modes

Vérification des flux d'exécution

- Vérification statique des connexions
- Utilisation de méthodes formelles

- p.ex.: réseaux de Petri
  - pour étudier l'absence d'interblocages provoqués par l'assemblage des composants
  - pour vérifier la définition correcte des données
  - détecter les sous-ensembles architecturaux jamais utilisés
  - …

Vérification de l'ordonnançabilité

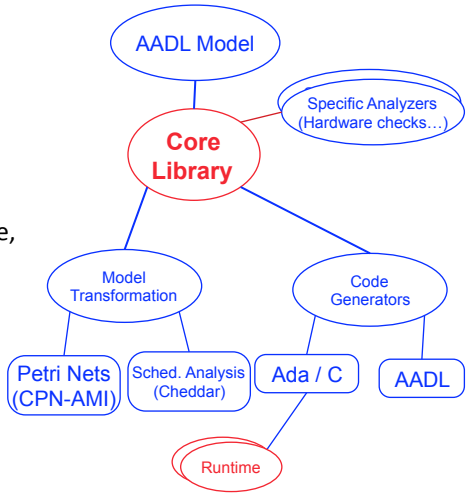
- Au niveau des processus et des processeurs
  - contrainte sur le nombre maximum de threads
  - calcul de l'ordonnancement
- Exploitation des propriétés AADL
  - protocole d'ordonnancement
  - temps d'exécution
  - communications entre les nœuds
- p.ex.: Cheddar

Génération de l'application

- Traduction des composants AADL en code source
  - types de données
  - Procédures
- Configuration de l'exécutif en fonction de l'architecture
  - nombre de threads
  - entrées/sorties
  - périodes
  - …
- Adéquation entre le code généré et la modélisation architecturale
  - les étapes de vérification et la génération de code doivent reposer sur les mêmes spécifications d'exécutif

Ocarina: un ensemble d'outils AADL

- Bibliothèque & outils pour manipuler AADL
  - Parseurs & afficheurs AADL
  - Vérification sémantique
- Générateurs de code
  - Ada/PolyORB
  - (Ada, C)/PolyORB-HI
  - Pour plusieurs plateformes (Native, LEON, ERC32)
- Configuration du support d'exécution
- Vérification & Validation
  - Réseaux de Petri
  - Ordonnancement (Cheddar)
- Un outil en ligne de commande pour automatiser ces tâches



PolyORB-HI: Intergiciel pour les systèmes critiques

- Contraintes temps réel dur spécifiques aux systèmes critiques:
  - Modèle de concurrence analysable : Profil Ravenscar
  - Restrictions du langage de programmation pour les systèmes critiques
    - Encore plus restrictif que le profil Ravenscar
  - Pas d'allocation dynamique ni d'orienté objet
- PolyORB-HI: un support d'exécution AADL
  - Supporte les constructions AADL
    - Threads périodiques et sporadiques, données, etc.
  - Configure automatiquement à partir du modèle AADL
    - Ressources calculées et allouées statiquement
    - Pas d'intervention requise de la part de l'utilisateur
  - Occupe une faible taille en mémoire
    - Toute la valeur ajoutée est dans la phase de génération de code
  - Contribuer à la thématique des "usines à intergiciels"

Conclusions

- Modélisation concrète
  - dernière phase avant la génération/déploiement du système
  - précision dans la modélisation
  - exploitation pour la vérification et la génération de prototypes
  - possibilités de corrections sur l'architecture
- AADL offre une grande souplesse de modélisation
  - degré de modélisation selon les besoins
  - peut être utilisé comme langage fédérateur
    - exploitation par plusieurs outils différents
- http://aadl.telecom-paristech.fr/ocarina
- http://aadl.telecom-paristech.fr/polyorb-hi
- http://beru.univ-brest.fr/~singhoff/cheddar/index-fr.html
- http://www-src.lip6.fr/logiciels/mars/CPNAMI/