Les langages SDL et MSC

Jean-Philippe Babau

INSA Lyon

Introduction

- Modélisation
  - description abstraite
  - spécification, implantation
- Validation / vérification
  - propriété du cahier des charges
  - propriétés applicatives (absence de deadlock, …)
  - analyse de performance
- Langages formels
  - sémantique non ambiguë
  - LOTOS, Estelle, SDL
- Techniques de validation
  - Simulation (exhaustive, aléatoire, test)
  - Preuves (logiques temporelles, model-checking, on-the-fly)

Plan

- Les langage SDL et MSC
  - Principes
- Vérification, tests
- Génération de code
- UML / SDL

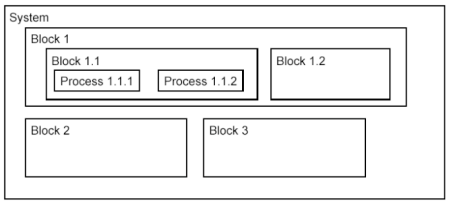
SDL : Specification and Description Language

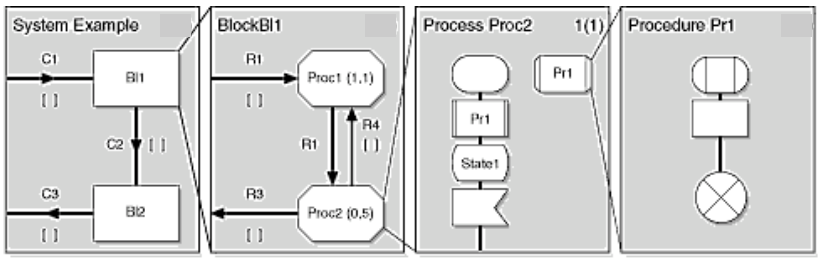
- Langage standard
  - Sémantique définie
  - Langage des télécom (ITU)
  - Langage normalisé (SDL 88, SDL 92, **SDL 96**, SDL2000)
- Principes
  - Langage graphique et textuel
  - Machines à états finis communicantes
- Outils
  - ObjectGéode, **RTDS** de PragmaDev, **Tau** de Télélogic
  - Edition
  - Vérification/Validation/test
    - MSC
    - observateurs
  - Génération de code

Principes généraux

- Décomposition arborescente
  - *system, block*, *substructure*, *process, procedure*
- Communication
  - Canaux de communication (canaux et routes)
  - Signaux avec paramètres optionnels
- Comportement dans les process et les procédures
- Typage des données (ADT : Abstract Data Type)

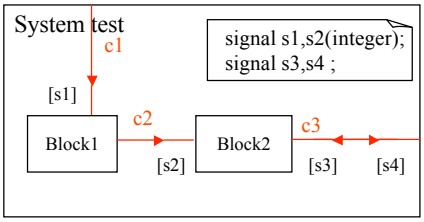
Structure





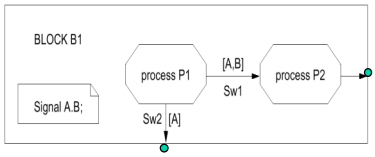
Le *system*

- Structure
  - *block*, canaux de communication
- Déclaration des signaux, des types



Le *block*

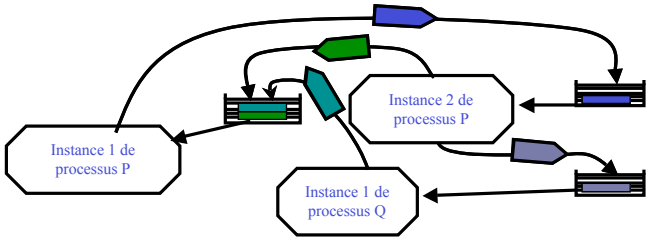
- Décomposition arborescente
  - *substructure*
- Machine à état fini
  - *process*
  - 1 ou plusieurs instances, un *PID* par instance
    - (i,j) : création statique de *i* instances, et *j* nb max de création dynamique d'instances
  - Exécution parallèle des *process*
- Communication par les *routes*
- Déclarations
  - Signaux internes
  - *procedure,* types



La communication

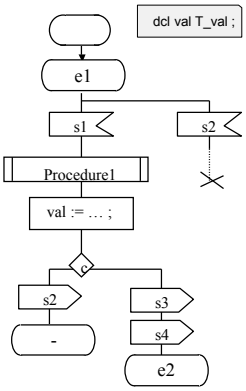
-Communication asynchrone par signal

- Le *signal* véhicule des paramètres
- Emis par une instance de process
- Destinataire
  - une instance ou toutes les instances
  - un seul consommateur (indéterministe)
  - gestion en FIFO sauf si signal prioritaire (*priority*)



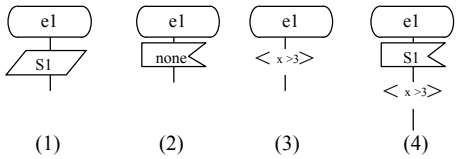
Description du comportement des *process*

- Etat
  - Un état de départ obligatoire
  - Pas d'état hiérarchique
  - Pas d 'états concurrents
  - Un état de fin
- Transition
  - Signal ou garde déclenchant
  - Corps
    - Appel de *procedure*, *task*
    - Alternatives
    - Émission de signaux
  - Retour dans un état
    - Identique ou différent
- Hypothèse RTC « Run To Completion »
  - i-e transition non préemptible



Description des *process*

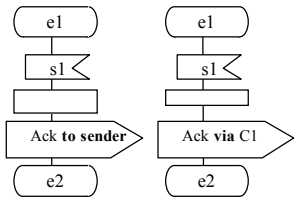
- Transition
  - (1) un signal non attendu est perdu sauf si sauvegarde explicite (*save*) du signal
  - (2) transition spontanée
  - (3) transition continue
  - (4) garde
    - (*sauve s1 tant que x<3*)

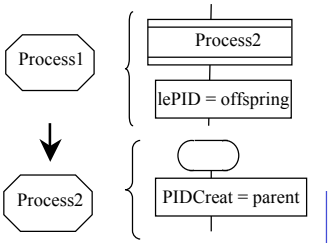


- Facilité d'écriture
  - Tous les états, tous les signaux *

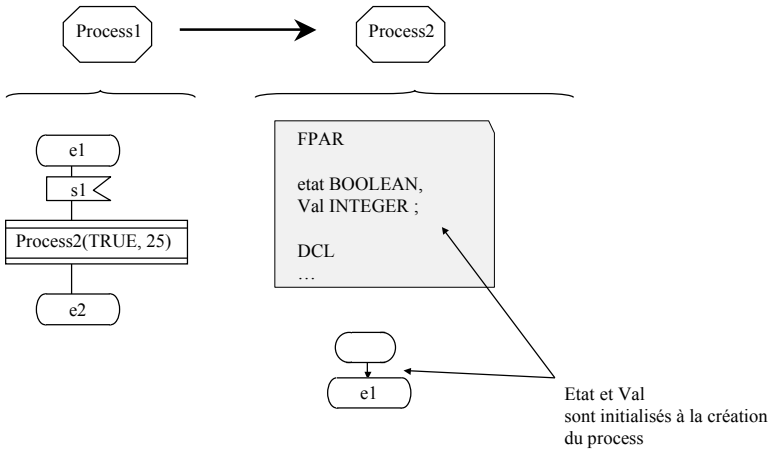
Gestion des identifiants de process

- PID
  - Type : *pid*
  - Identifiant du process : variable *self*
- communication
  - Réception
    - pid de l'émetteur du dernier signal reçu : *sender*
    - Filtrage en réception : *from*
  - Émission
    - Destinataire explicite : *to*
    - Canal explicite : *via*
- Création dynamique
  - dernier process créé : variable *offspring*
  - créateur : variable *parent*





Paramètres formels de process



Les données

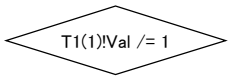
- Type Abstrait de données
  - Type prédéfinis (*INTEGER*, *REAL*, *NATURAL*, *BOOLEAN*, *CHARACTER*, *CHARSTRING*, *DURATION*, *TIME*, *Pid*, *ARRAY*, STRUCT, POWERSET)
  - *SYNONYM, SYNTYPE, NEWTYPE, LITERALS*
- Données encapsulées
  - Propre à chaque instance de process
  - Déclaration : *dcl* val Type *;*

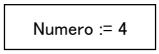
```
SYNONYM NbNodes = 10;

SYNTYPE Index = INTEGER
CONSTANTS 1:NbNodes
ENDSYNTYPE ;

NEWTYPE champ
STRUCT
        Num INTEGER;
        Val INTEGER;
DEFAULT (. 0, 0 .);
ENDNEWTYPE;

NEWTYPE Trame ARRAY(Index, champ)
ENDNEWTYPE;
```

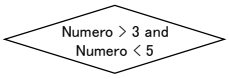
```
DCL numero integer := 0 ;
DCL T1 Trame;
```

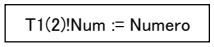
Les données

- Les opérateurs
  - Comparaison = /= > <= …
  - opérateurs logiques or and not xor
  - Opérateurs numériques + - * /
  - Affectation :=
  - tableau make! extract! modify!
    - À utiliser uniquement dans les axiomes



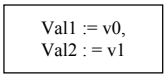




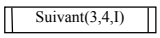


Les actions

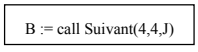
- Task
  - Affectations



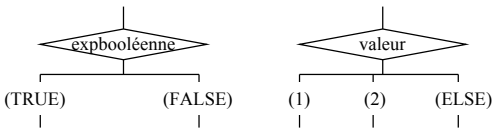
- Appel de procédure
  - Local



  - Distant
    - Émission implicite d'un signal
    - état implicite d'attente
    - RPC

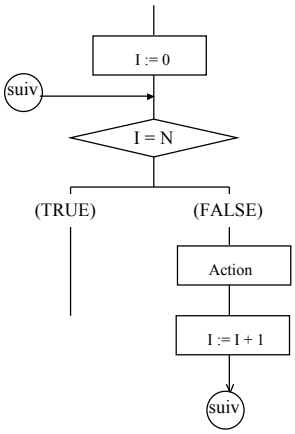


- Conditionnelles

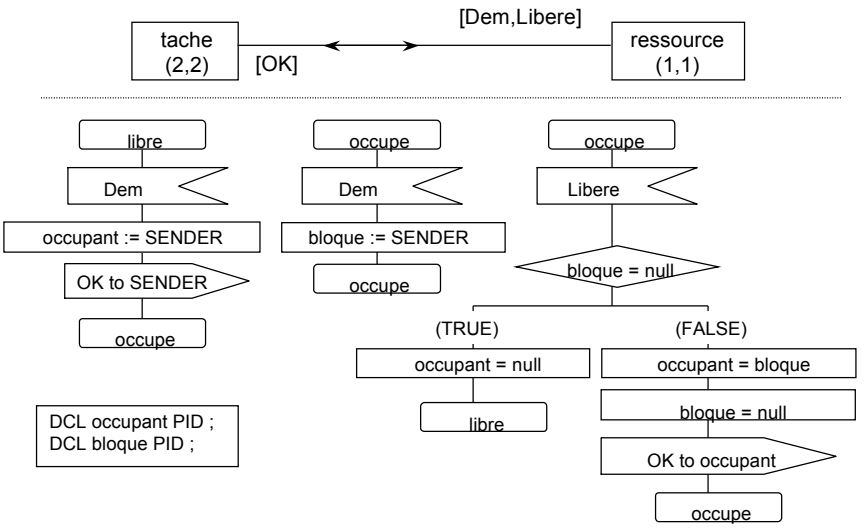


Les actions

- Labels
  - Utiles pour les boucles
  - goto : à éviter



SDL exemple

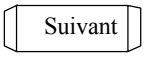


SDL exemple

- si Libere quand libre : ?
- si re-demande : ?
- si libere par mauvaise tâche : ?
- état occupe / demande : ?
- si plusieurs tâches : ?

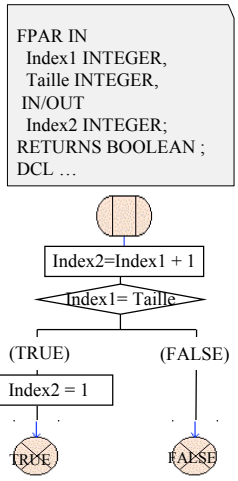
Description de procédures

- Déclaration



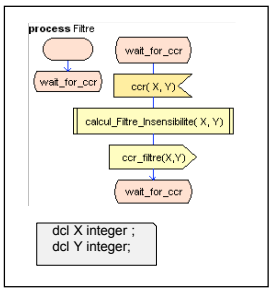
- Algorithme
  - Début
  - Fin
  - État

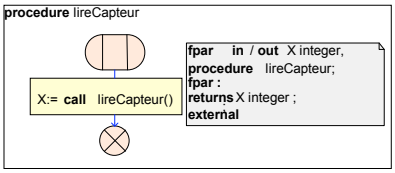
- Paramètres formels
  - Entrée : IN
  - Entrée/sortie : IN/OUT
  - Retour : RETURNS



Description de procédures

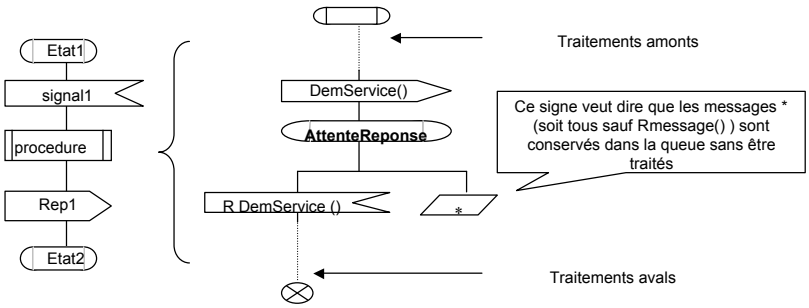
- Appel de procédure externe C
  - Signature
  - Simulation
    - Génération de code
    - Édition de lien





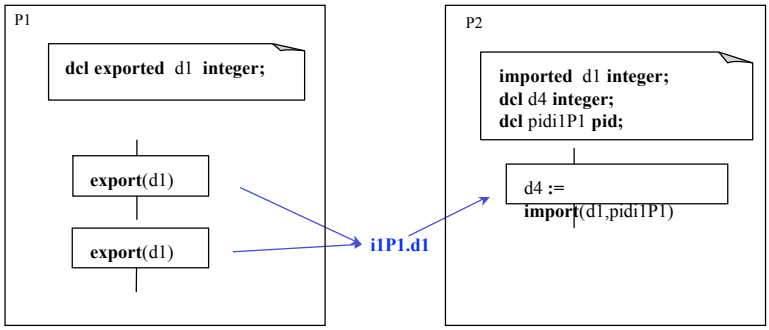
Description de procédures

- État implicite du process



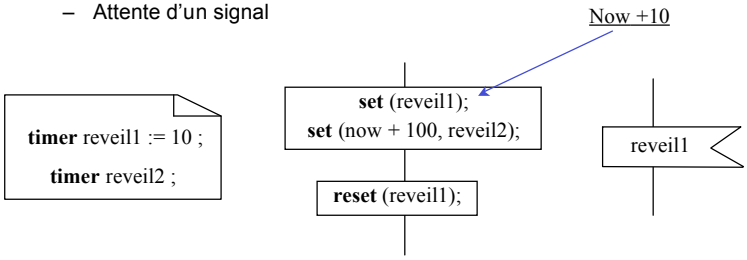
Les données partagées

- Modèles de partage
  - revealed/view
    - multi-lecteur/mono-écrivain, partage d'une donnée entre 2 processus d'un même bloc
  - export/import + remote
    - multi-lecteur/multi-écrivain,



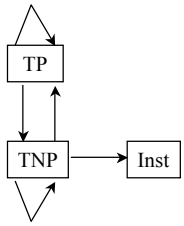
SDL et le temps

- Données
  - Deux types prédéfinis: *Time* et *Duration*
  - Estampille locale en utilisant *Now*
- *Timer* (temporisateur)
  - Déclaré et utilisé par un seul process
  - Actions : Armer (set), Arrêter (reset)
  - Attente d'un signal



SDL et les types

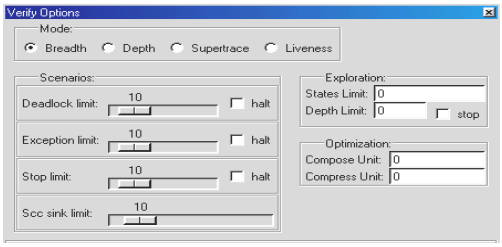
- Héritage simple

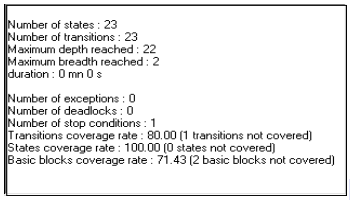


- Types paramétrés
  - Paramètre formel de contexte
    - System, block
      - Procédure, signal, synonyme, type de données
    - Process
      - Process, procédure, signal, synonyme, type de données
    - Procédure
      - Process, procédure, signal, synonyme, variable, réveil type de données
  - Type non paramétré
    - Instance de type non paramétré
- Spécialisation (*inherits*)
  - Ajout
  - Modification (*virtual / redefined*)
    - Block, processus, procedure, transition

Vérification (Objectgéode)

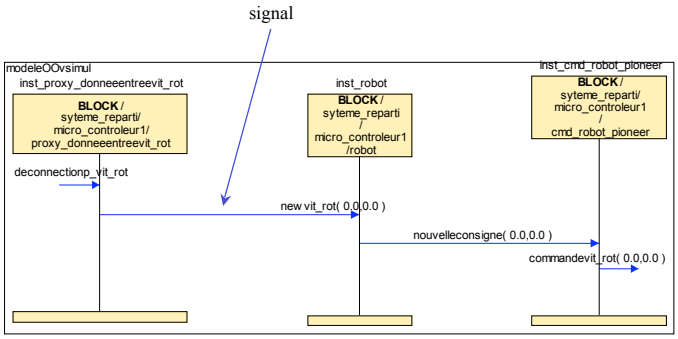
- Simulation
  - Point d'arrêt sur conditions (états, valeurs)
  - Mode pas à pas : mise au point
  - Mode aléatoire : test
  - Mode simulation exhaustive
    - Preuve sur les états
      - Si pas d'explosion
    - Tests si données (domaine d'entrée non parcouru)





Vérification avec (Objectgéode)

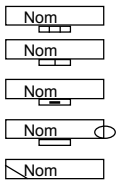
- Diagramme de séquence (formalisme MSC : Message Sequence Chart)
  - Spécification, génération lors de la simulation
  - Observateur



Message Sequence Charts (MSC)

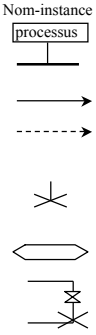
Opérateurs de scénarii

- Parallèle
- Ou
- Séquence
- Répétition
- Scénario terminal



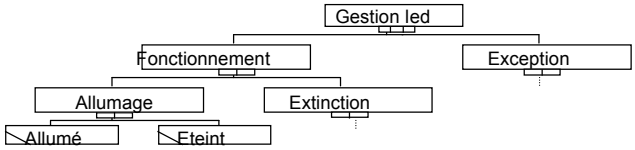
Scénario terminal

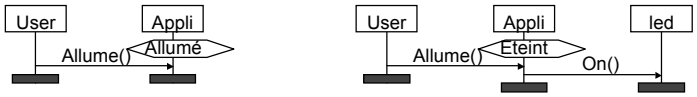
- Instance
- signal
- Création d'objets
- Fin d'instance
- Etat
- Timers (set, reset, expiration)



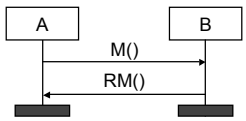
MSC exemples

Exemple



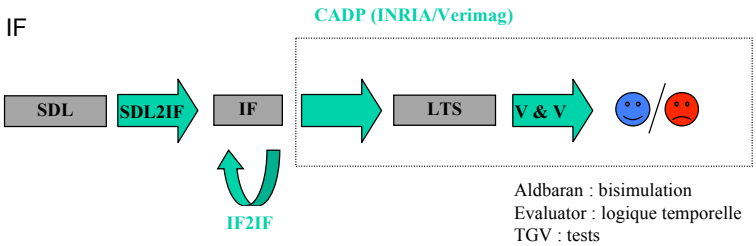


Appel synchrone



Vérification formelle

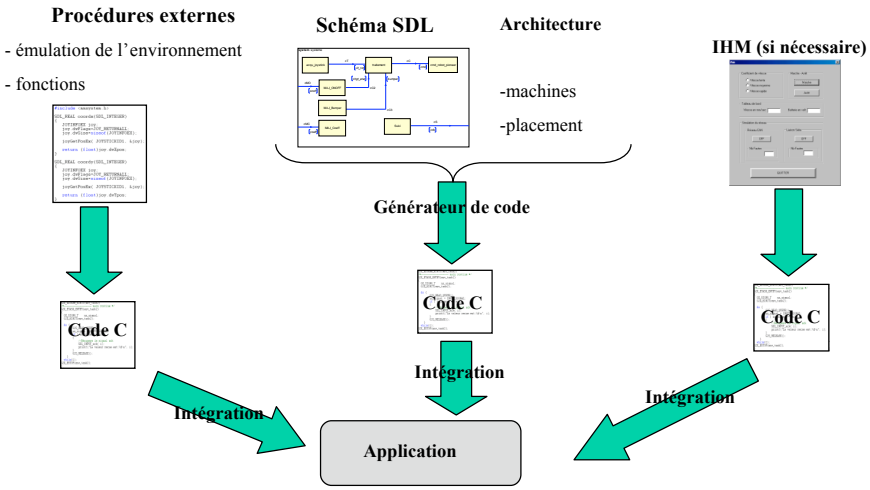
- SDL -> UPPAAL
- IF



Aldbaran : bisimulation
Evaluator : logique temporelle
TGV : tests

- Explosion
  - Réduction, interprétation

Génération de code : ObjectGéode



ObjectGéode

- Machine virtuelle qui s'appuie sur l'OS (WIN32, UNIX, RTOS)
  - tâches, boîtes aux lettres, sémaphores, socket …
- Architecture
  - placement statique des process SDL sur des machines
  - mono processus au sens OS
  - communication par socket
  - Une tâche par machine en attente de signaux
- Comportement
  - États et transitions entièrement gérés
  - Actions sur les transitions en langage cible (C,C++)

ObjectGéode

- Concurrence
  - Une tâche pour le système, par block, par process, par instance de process (selon l'OS)
  - Granularité variable
- Une tâche dédiée aux timers
  - Gère une structure de données de tous les timers en attente
  - set (timerID,temps,liste de PIDs, paramètres)
  - reset(timerID)
  - timeout : « plus faible » timer, envoi de messages aux instances en attente
  - Unité de temps
- Affectation des priorités
  - Un niveau de priorité par défaut (200 sous VRTX), modifiable

UML / SDL

- SDL 2000 / UML 2.0
  - SDL2000
    - association : commentaire, état composite
    - pas d'outil
  - UML 2.0
    - UML standard objet, Télélogic dans l'OMG
    - langage d'action compatible avec SDL
    - Diagramme de séquence : les MSC
- UML -> SDL
  - Restrictions
    - Pas d'héritage multiple, pas d'associations binaires, pas de contraintes OCL
  - Classe -> process / type de données
  - États -> remise à plat
  - Relation
    - Héritage -> type
    - Association -> pid
- SDL : une sémantique pour UML

Utilisation de SDL

- Langage pour la description de systèmes distribués discrets
- Implémentation
  - Précisions sur "comment le modèle s'exécute"
  - Respect de la sémantique SDL
  - Optimisation de la QoS
- Liens avec d'autres formalismes
  - Techniques formelles de vérification
  - Systèmes continus : environnement du système
- Méthodes de développement
  - Modèles, schémas types
  - Formalisation
  - Prise en compte de tous les aspects

Références

- *Normes SDL et MSC*
  - ITU-T « *Recommendation Z.100, Specification and Design Language (SDL)* », 1996
  - ITU-T « *Recommendation Z.100, Specification and Design Language (SDL2000)* », 1999
  - ITU-T « *Recommendation Z.120, Diagrammes des séquences de messages (MSC)* », 1999
- *Présentation du langage*
  - *SDL, Modélisation de protocoles et systèmes réactifs, Zoubir Mammeri , Ed Hermès Science Europe, 2000, ISBN: 2-7462-0166-6*