**Simulation d’un protocole entre un processus et**

**un environnement composé de 1 acteur**

Version : oneProc\_with\_context\_Async.fcr

Exploration avec OBP Explorer

Ph. Dhaussy

18 dec 15

**Résumé**

Cette note relate la modélisation d’un processus (Fiacre) en interaction avec un environnement composé de 1 acteur (modélisé en CDL).

Elle montre les possibilités d’observation du comportement du circuit et de vérification de propriétés.

**Sommaire**

[1. Architecture du circuit de calcul 1](#_Toc438205516)

[2. Modélisation des scénarios 2](#_Toc438205517)

[2.1. Scénario : cdl\_dev1 2](#_Toc438205518)

[2.2. Scénario : cdl\_dev1\_error1 3](#_Toc438205519)

[3. Modélisation de propriétés à vérifier 4](#_Toc438205520)

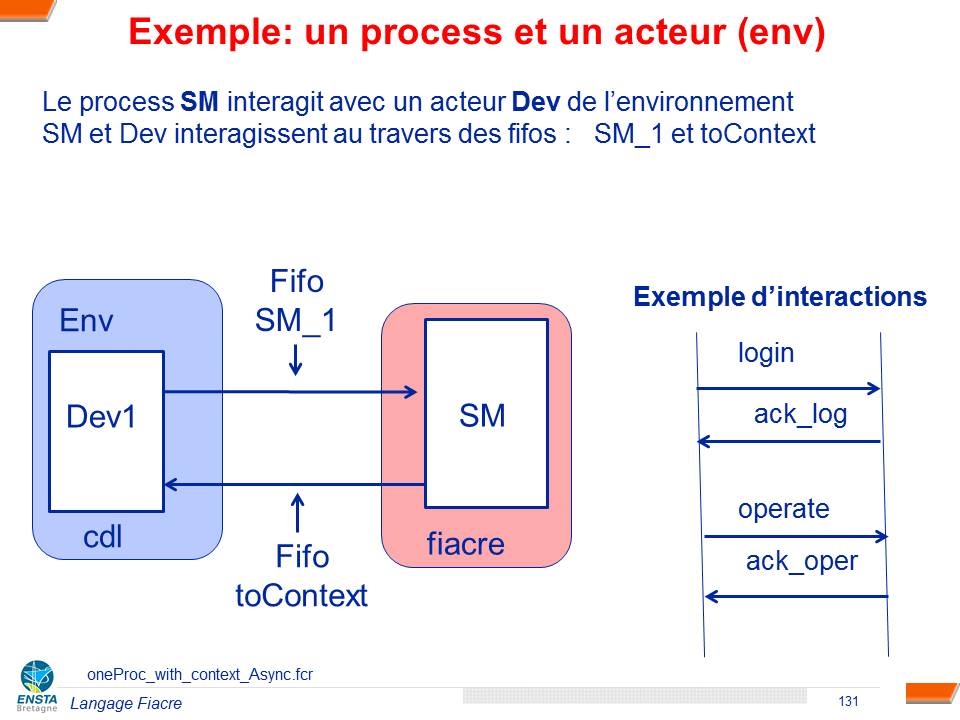
[2. Programme Fiacre 5](#_Toc438205521)

[3. Programme CDL 8](#_Toc438205522)

# Architecture du circuit de calcul

Le principe de l’architecture est (figure 1) :

* Un processus **SM** interagit avec un acteur **Dev1** de l’environnement.
* SM et Dev1 interagissent au travers de deux fifos :
  + **SM\_1** : pour la communication vers SM
  + **toContext** : pour la communication vers l’environnement



**Figure 1 : architecture du circuit de calcul**

Les interactions entre l’environnement s’effectuent de la manière suivante  :

* Le device Dev1 initie une interaction en envoyant un message *login*. Ensuite, il peut envoyer un ou plusieurs messages *operate* qui modélise une opération à exécuter par SM. L’interaction se termine par l’envoi d’un message *logout*. Un nouvel *operate* ne sera possible que suite à l’envoi d’un nouveau *login*.
* Dev1 envoit le message *login* vers SM qui répond ack\_log ou nack\_log.
* Dev1 envoit le message *operate* vers SM qui répond *ack\_oper* ou nack\_oper.

Les programmes Fiacre et CDL sont donnés en Annexe.

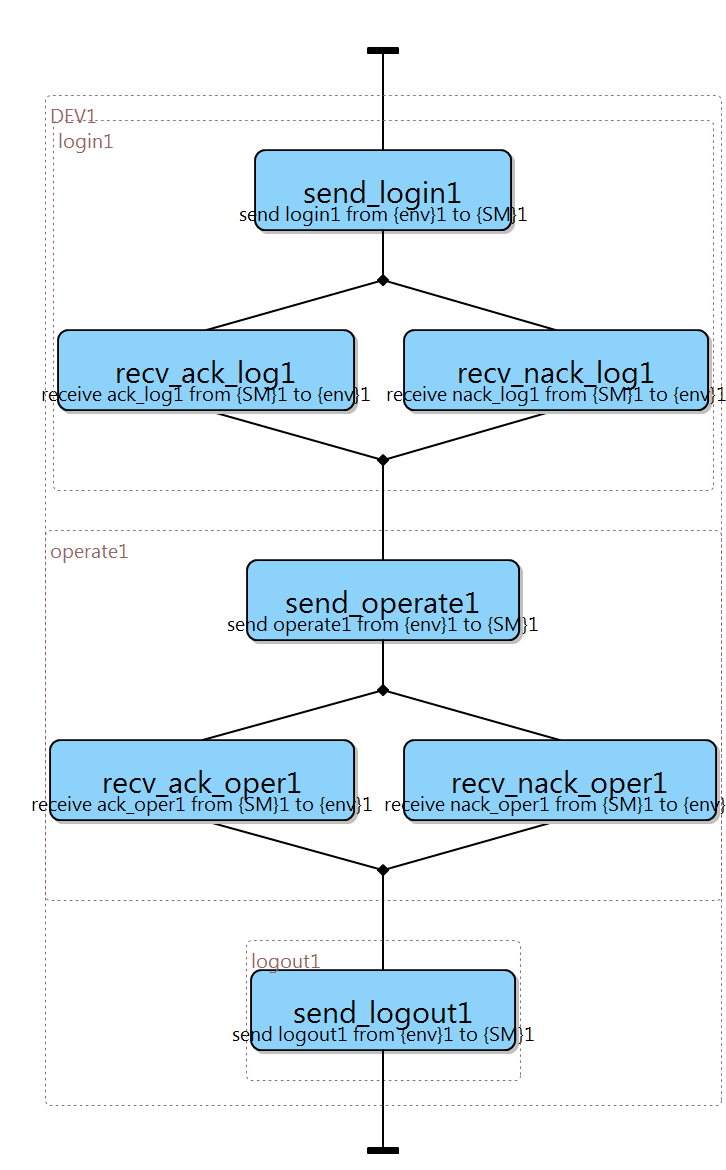
# Modélisation des scénarios

Exemples de scénarios pour tester le modèle.

## 2.1. Scénario : cdl\_dev1

Un seul acteur dans l’environnement : DEV1 (mode nominal)

Le comportement de DEV1 est spécifié par l’activité DEV1.



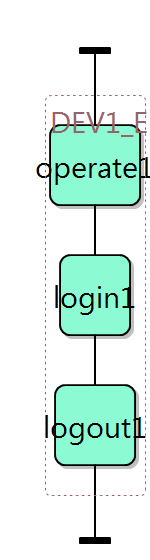
Code CDL :

activity DEV1 is { login ; operate; logout }

## 2.2. Scénario : cdl\_dev1\_error1

Un seul acteur dans l’environnement : DEV1\_Error1 (Envoi d’operate avant un login).

Le comportement de DEV1\_Error1 est spécifié par l’activité DEV1\_Error1.



Code CDL :

activity DEV1\_Error1 is { operate1; login1; logout1 }

# Modélisation de propriétés à vérifier

Selon les cas, les exigences peuvent tester le comportement soit de l’environnement, soit du processus SM.

**Quelques exemples :**

Req1 : Un *login* est suivi par un *logout* avant le *login* suivant (non réception de 2 *login* de suite) (test du comportement de l’environnement).

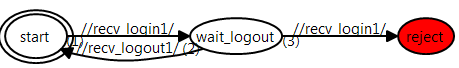
Req2 : Un *ack\_log* est précèdé par un *login* (test de SM).

Req3 : Un *operate* n’est pas suivi par un 2ème *operate* sans l’émission précédemment d’un *ack\_oper* ou d’un n*ack\_oper* (test du comportement de l’environnement).

Req4 : Un *operate* n’est accepté que si le device associé est déjà logué (un *login* a été émis précédemment) (test de SM).

Modélisation de Req 1

Req1 peut se modéliser par l’observateur suivant :



Spécification CDL :

property pte\_login1\_logout1 is

{

// ---- nominal ----

start -- / / recv\_login1 / -> wait\_logout;

wait\_logout -- / / recv\_logout1 / -> start;

// ---- errors ----

wait\_logout -- / / recv\_login1 / -> reject

}

# Programme Fiacre

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

oneProc\_with\_context\_Async.fcr

Ph. Dhaussy Ensta-Bretagne v 28 dec 11

--------------------------------------------

Un processus SM dans le systeme.

Un acteur dans l'environnement.

Communication asynchrone entre SM et l'environnement.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const TAILLE\_FIFO : nat is 10

/\*--------------------------------------------------

DECLARATIONS DES STRUCTURES

----------------------------------------------------\*/

type t\_signal\_type is union

NUL

| LOGIN

| LOGOUT

| ACK\_LOG

| NACK\_LOG

| OPERATE

| ACK\_OPER

| NACK\_OPER

| FIN

end union

type T\_SIGNAL is record

signal\_type : t\_signal\_type /\*caracterise l'id du signal \*/

end

/\*--------------------------------------------------

FIFO

----------------------------------------------------\*/

/\* pour les fifo partagees \*/

type fifo is queue TAILLE\_FIFO of T\_SIGNAL

/\*----------------------------------------------------

Declarations de constantes pour les messages

------------------------------------------------------\*/

/\* Env <---> SM \*/

const login : T\_SIGNAL is {signal\_type = LOGIN}

const logout : T\_SIGNAL is {signal\_type = LOGOUT}

const ack\_log : T\_SIGNAL is {signal\_type = ACK\_LOG}

const nack\_log : T\_SIGNAL is {signal\_type = NACK\_LOG}

const operate : T\_SIGNAL is {signal\_type = OPERATE}

const ack\_oper : T\_SIGNAL is {signal\_type = ACK\_OPER}

const nack\_oper : T\_SIGNAL is {signal\_type = NACK\_OPER}

const msg\_fin : T\_SIGNAL is {signal\_type = FIN}

/\*================================================

SM

===============================================\*/

process SM ( &input : read write fifo,

&to\_context : read write fifo

)

is

states

waitSignal,

sendAckLog,

sendAckOper,

Fin

var

received : T\_SIGNAL, /\* msg recu \*/

sent : T\_SIGNAL /\* msg envoye \*/

init

received := {signal\_type = NUL};

sent := {signal\_type = NUL};

to waitSignal

/\*-----------------------------------------------------------

receive soit :

LOGIN,

OPERATE,

ACK\_RIGHTS,

NACK\_RIGHTS,

FIN

------------------------------------------------------------\*/

from waitSignal

//------- attente et reception d'un message -------

if (empty input) then loop end;

received := first input;

input := dequeue (input);

case (received.signal\_type) of /\*--- message recu ? ---\*/

LOGIN ->

/\* LOGIN recu \*/

to sendAckLog

| OPERATE ->

/\* OPERATE recu \*/

to sendAckOper

| FIN ->

/\* FIN recu \*/

to Fin

end case;

/\* autre msg recu \*/

loop

/\*------------------------------------------------

send ACK\_LOG a Dev

-------------------------------------------------\*/

from sendAckLog

if (full (to\_context)) then loop end;

sent.signal\_type := ACK\_LOG;

// sent.signal\_type := NUL;

to\_context := enqueue (to\_context, sent);

to waitSignal // se remettre en attente d'un message

/\*------------------------------------------------

send ACK\_OPER a Dev

-------------------------------------------------\*/

from sendAckOper

if (full (to\_context)) then loop end;

// sent.signal\_type := ACK\_OPER;

sent.signal\_type := NUL;

to\_context := enqueue (to\_context, sent);

to waitSignal // se remettre en attente d'un message

/\*------------------------------------------------

Fin

-------------------------------------------------\*/

from Fin

loop

/\*===============================================

Composant

==============================================\*/

component main is

var

SM\_1 : fifo, // obligatoire : prend le nom du process

toContext : fifo //Non necessaire si pas de cdl

init

SM\_1 := {||};

toContext := {||}

par

SM (&SM\_1, &toContext)

end

main

# Programme CDL

//==========================================================

// oneProc\_with\_context\_Async.cdl

//

// Ph. Dhaussy Ensta-Bretagne v 28 dec 11

//===========================================================

event send\_login is {send login to {SM}1}

event recv\_ack\_log is {receive ack\_log from {SM}1}

event recv\_nack\_log is {receive nack\_log from {SM}1}

event send\_operate is {send operate to {SM}1}

event recv\_operate is {receive operate from {env}1 to {SM}1}

event recv\_ack\_oper is {receive ack\_oper from {SM}1}

event recv\_nack\_oper is {receive nack\_oper from {SM}1}

event send\_msg\_fin is {send msg\_fin to {SM}1}

event recv\_msg\_fin is {receive msg\_fin from {env}1 to {SM}1}

//===========================================================

// déclaration des predicats

//===========================================================

predicate SM\_fin is { {SM}1@Fin }

//===========================================================

// déclaration des evenements

//===========================================================

event evt\_sm\_fin is { SM\_fin becomes true}

//===========================================================

// déclaration des proprietes

//===========================================================

property pte\_fin is {

start -- / / evt\_sm\_fin / -> success

}

//===========================================================

// déclaration des interactions

//===========================================================

// Dev <---> SM

activity login is { event send\_login;

{ event recv\_ack\_log [] event recv\_nack\_log } }

activity loginAck is { event send\_login; event recv\_ack\_log }

activity loginNack is { event send\_login; event recv\_nack\_log }

activity loginBis is { loginAck [] loginNack }

activity operate is { event send\_operate;

{ event recv\_ack\_oper [] event recv\_nack\_oper } }

activity fin is { event send\_msg\_fin }

//===========================================================

// comportements de l'environnement

//===========================================================

activity DEV is

{

login;

operate;

fin

}

activity DEVBis is

{

loginBis;

operate;

fin

}

//===========================================================

// Scenarios CDL

//===========================================================

cdl cdl\_1 is {

properties pte\_fin

main is { DEV }

}

cdl cdl\_1Bis is {

properties pte\_fin

main is { DEVBis }

}