Simulation d'un protocole entre un processus et un environnement composé de 2 acteurs

Philippe Dhaussy

Lab-Sticc, ENSTA Bretagne.
2, rue François Verny. 29806 Brest France
philippe.dhaussy@ensta-bretagne.fr

Résumé

Cette note relate un exemple de modélisation d'un protocole d'interaction entre un système (simulé par un processus) et un environnement composés d'acteurs. Les acteurs se logguent sur le système et demande le déclenchement d'opérations.

La note illustre la formalisation des scénarios et des propriétés CDL montrant les possibilités d'observation et de validation du comportement du protocole.

Les programmes Fiacre et CDL sont donnés en Annexe.

Table des matières

1 Architecture du protocole

On veut modéliser un protocole d'interaction entre un système (simulé par un processus SM) et un environnement (simulé par des acteurs Dev). Les acteurs déclenchent des opérations ($operate$) sur le système. Au préalble, les acteurs doivent se loguer sur le système ($login$) avant de faire leur requête d'opérations. Un acteur clôt son interaction en se déloggant ($logout$).

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

- Un processus SM interagit avec deux acteurs $Dev1$ et $Dev2$ de l'environnement.
- $SM, Dev1$ et $Dev2$ interagissent au travers de deux fifos :
  - SM_1 : pour la communication vers SM
  - $toContext$: pour la communication vers l'environnement

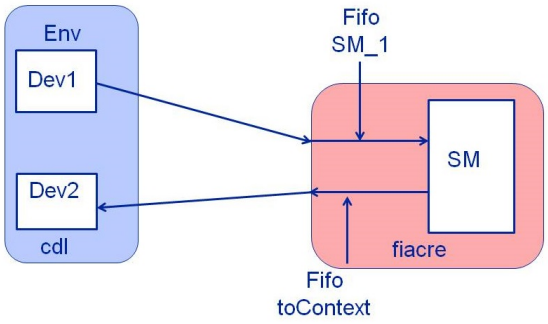

Figure 1. Architecture.

Les interactions entre l'environnement s'effectuent de la manière suivante (figure 2) :

- Un device (Dev1 ou Dev2) 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 (resp. Dev2) envoit le message $login1$ (resp. $login2$) vers SM qui répond ack_log1 (resp. ack_log2) ou $nack\_log1$ (resp. $nack\_log2$)
- Dev1 (resp. Dev2) envoit le message $operate1$ (resp. $login2$) vers SM qui répond ack_oper1 (resp. ack_oper2) ou $nack\_oper1$ (resp. $nack\_oper$)

2 Modélisation des scénarios

Quelques exemples de scénarios pour tester le modèle sont décrits ci-dessous.

2.1 Scénario : *cdl_dev1*

Un seul acteur dans l'environnement : DEV1 (mode nominal) Le comportement de $DEV1$ est spécifié par l'activité $DEV1$ (figure 3).

Le code CDL est :

```
activity DEV1 is  { login1; operate1; logout1 }
cdl cdl_dev1 is
{
```

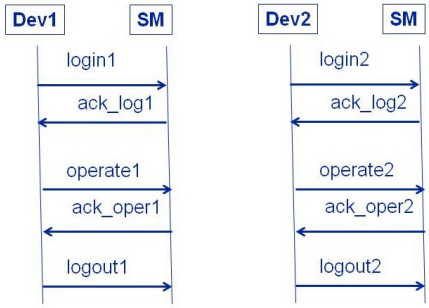

Figure 2. Exemples d'interactions.

```
    main is { DEV1 }
}
```

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 $DEV1\_Error1$ (figure 4).

Le code CDL est :

```
activity DEV1_Error1 is  { operate1; login1; logout1 }
cdl cdl_dev1_error1 is
{
    main is { DEV1_Error1 }
}
```

2.3 Scénario : *cdl_dev1_error2*

Un seul acteur dans l'environnement : $DEV1\_Error2$ (Envoi d'*operate* avant un *logout*). Le comportement de $DEV1\_Error2$ est spécifié par l'activité $DEV1\_Error2$ (figure 5).

Le code CDL est :

```
activity DEV1_Error2 is  { login1; logout1; operate1 }
cdl cdl_dev1_error2 is
{
    main is { DEV1_Error2 }
}
```

2.4 Scénario : *cdl_dev1_error1_dev2*

Le code CDL est :

```
activity DEV1_Error1 is  { operate1; login1; logout1 }
activity DEV2        is  { login2; operate2; logout2 }
cdl cdl_dev1_error1_dev2 is
{
    main is { DEV1_Error1 || DEV2 }
}
```

2.5 Scénario : *cdl_dev1_error2_dev2*

Le code CDL est :

```
activity DEV1_Error2 is  { login1; logout1; operate1 }
activity DEV2        is  { login2; operate2; logout2 }
cdl cdl_dev1_error1_dev2 is
{
    main is { DEV1_Error2 || DEV2 }
}
```

2.6 Scénario : *cdl_dev1_3xDev2*

Le code CDL est :

```
activity DEV1 is  { login1; operate1; logout1 }
activity DEV2 is  { login2; operate2; logout2 }
cdl cdl_dev1_3xDev2 is
{
     main is { DEV1 || DEV2 || DEV2 || DEV2 }
}
```

3 Modélisation de propriétés à vérifier

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

Ci-dessous, quelques exemples.

3.1 Req 1

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).

Req1 peut se modéliser par l'observateur suivant :

Code CDL :

```
property pte_login1_logout1 is
{
// ---- nominal ----
start        -- / / recv_login1  / -> wait_logout;
wait_logout  -- / / recv_logout1 / -> start;
// ---- errors ----
wait_logout  -- / / recv_login1  / -> reject
}
```

3.2 Req 2

Req2 : Un *ack_log* est précèdé par un *login* (test de SM).

Req2 peut se modéliser par l'observateur suivant :

```
property pte_login1_precedes_acklog1 is
{
// ---- nominal ----
start          -- / / send_login1     / -> wait_ack_log;
wait_ack_log   -- / / recv_ack_log1  / -> start;
// ---- errors ----
start          -- / / recv_ack_log1  / -> reject;
wait_ack_log   -- / / recv_nack_log1 / -> reject
}
```

3.3 Req 3

Req3 : Un *operate* n'est pas suivi par un 2ème *operate* sans l'émission précédemment d'un *ack_oper* ou d'un $nack_oper$ (test du comportement de l'environnement).

Req3 peut se modéliser par l'observateur suivant :

```
property pte_not_2_operate1_without_ack_oper1 is
{
// ---- nominal ----
start          -- / / send_operate1    / -> wait_ack_oper;
wait_ack_oper  -- / / recv_ack_oper1   / -> start;
wait_ack_oper  -- / / recv_nack_oper1  / -> start;

// ---- errors ----
start          -- / / recv_ack_oper1  / -> reject;
start          -- / / recv_nack_oper1 / -> reject;
wait_ack_oper  -- / / send_operate1   / -> reject
}
```

3.4 Req 4

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

Req4 peut se modéliser par l'observateur suivant :

```
property pte_operate1_if_Logged is
{
// ---- nominal ----
start          -- /                   / send_operate1   / -> wait_ack_oper;
wait_ack_oper  -- / SM_Dev1_logged    / recv_ack_oper1  / -> start;
wait_ack_oper  -- / SM_Dev1_Notlogged / recv_nack_oper1 / -> start;

// ---- errors ----
wait_ack_oper  -- / SM_Dev1_Notlogged / recv_ack_oper1  / -> reject;
wait_ack_oper  -- / SM_Dev1_logged    / recv_nack_oper1 / -> reject
}
```

Références

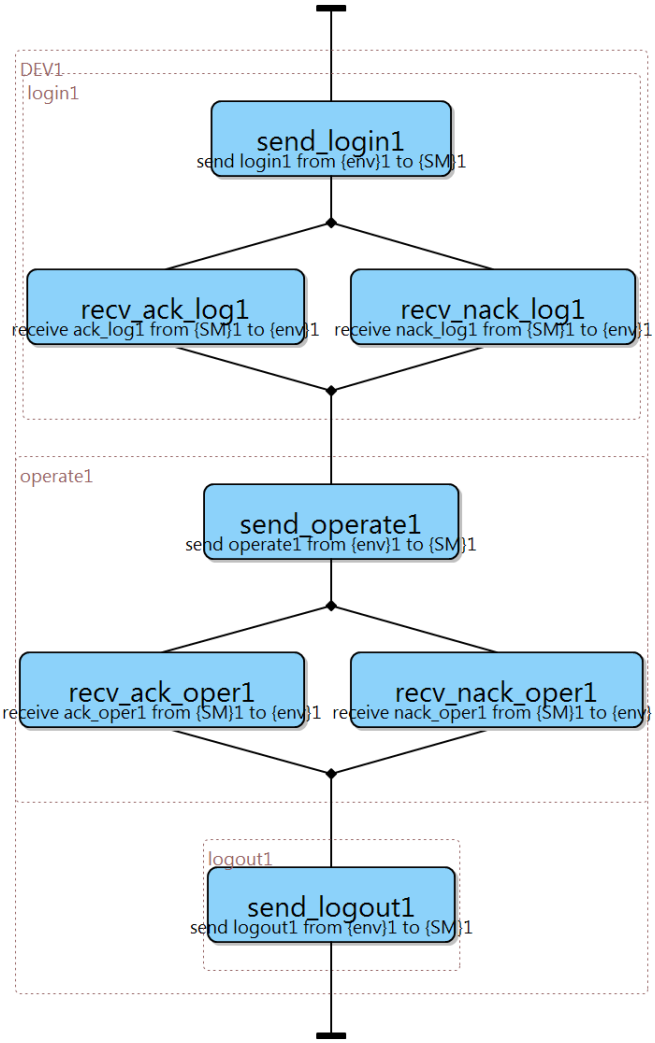

Figure 3. Activité DEV1.

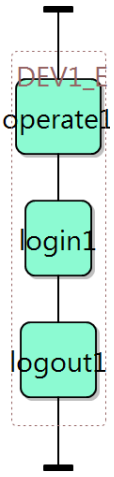

Figure 4. Activité $DEV1\_Error1$.

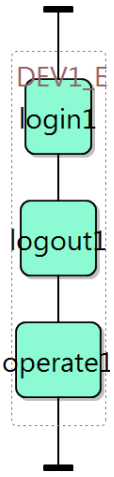

Figure 5. Activité $DEV1\_Error2$.

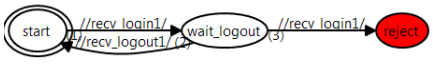

Figure 6. Req1 : Observateur pour $pte\_login1\_logout1$.

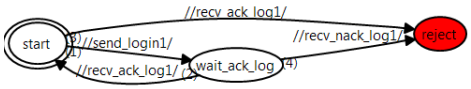

FIGURE 7. Req1 : Observateur pour $pte\_login1\_precedes\_acklog1$.

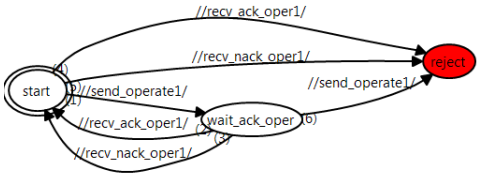

FIGURE 8. Req1 : Observateur pour $pte\_not\_2\_operate1\_without\_ack\_oper1$.

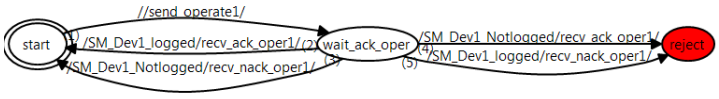

FIGURE 9. Req1 : Observateur pour $pte\_operate1\_if\_Logged$.