Etude B du système d'entrées-sorties SCSI-2

Didier Bert

2 mars 1999

Le document de référence de ce travail est la note [GM98]. Cependant, comme l'a noté Jean-Raymond Abrial [Abr98], il est nécessaire de réécrire certaines parties du cahier des charges pour avoir une vision abstraite du problème pendant l'étude système. Dans les paragraphes 1.1, 1.2 et 1.3, 1.4, on construit le développement global abstrait pour arriver à la description du paragraphe 1.5.

1 Etude système et développement de la spécification

1.1 Première étape de la modélisation : que doit faire le système d'entrées-sorties

1.1.1 Principe du système

Le système est composé d'un contrôleur CTR et de disques DSK. Le contrôleur envoie des requêtes REQ aux disques et les disques satisfont les requêtes suivant une certaine politique de priorité. L'idée du système est donc que les disques satisfont des requêtes. Dans ce genre de problème, on cherche à savoir si les requêtes sont toutes satisfaites et de quelle manière. En s'appuyant sur la modélisation présentée en [AM98], on peut proposer la première modélisation qui décrit la satisfaction des requêtes, indépendamment du nombre de disques et même de la présence du contrôleur. L'histoire des requêtes satisfaites est décrite par :

$$l_0 \in \mathbb{N} \mapsto REQ$$

Dans cette notation, $l_0(t)$ est la requête qui a été satisfaite au "moment" t. On impose la contraite que le domaine de l_0 est fini (c.a.d. le nombre de requêtes traitées est fini) :

$$\mathsf{dom}(l_0) \in \mathbb{F}(\mathbb{N})$$

Enfin, on note c_0 le moment de la dernière requête satisfaite :

$$\begin{array}{l} c_0 \in \mathbb{N} \\ l_0 \neq \emptyset \Rightarrow \ c_0 = \mathsf{max}(\mathsf{dom}(l_0)) \end{array}$$

Le système est constitué d'un seul événement : *dsk_rec* qui signifie qu'un disque vient de traiter une requête à l'instant t.

SYSTEM
$SCSI2_0$
VARIABLES
l_0, c_0
INVARIANT
$l_0 \in \mathbb{N} \pfun REQ \ \wedge$
$\mathsf{dom}(l_0) \in \mathbb{F}(\mathbb{N}) \ \wedge$
$c_0 \in \mathbb{N} \ \wedge$
$l_0 \neq \emptyset \Rightarrow \ c_0 = \mathsf{max}(\mathsf{dom}(l_0))$
INITIALISATION
$l_0, c_0 := \emptyset, 0$
EVENTS
$dsk\_rec =$ /* traitement d'une requête */
ANY t, x WHERE
$t \in \mathbb{N} \ \wedge \ x \in REQ \ \wedge \ c_0 < t$
THEN
$c_0 := t \ ||$
$l_0 := l_0 \cup \{t \mapsto x\}$
END
END

Sur cette première modélisation, on pourrait montrer un certain nombre de propriétés sur l'écoulement du temps, comme cela est fait en [AM98]. Nous ne les reprenons pas ici.

1.1.2 Premier raffinement: requêtes envoyées

Dans ce premier raffinement (de la conception du système), on introduit les requêtes envoyées par le contrôleur. Il s'agit aussi d'une suite de requêtes r_1 qui sont instampillées par *le moment où elles seront satisfaites*. En effet, le bon moyen pour vérifier qu'elles seront satisfaites à un instant t dans le futur, est d'indiquer cet instant, ou une borne supérieure de cet instant. Les nouvelles variables du système sont les mêmes que précédemment (avec renommage pour la facilité de lecture) l_1 et c_1, plus la séquence envoyée par le contrôleur. Dans cette modélisation, on a donc :

$$l_1 \subseteq r_1$$

ce qui signifie que les requêtes satisfaites sont des requêtes qui ont été envoyées, et $r_1 - l_1$ représente les requêtes en attente d'être traitées. Comme c_1 est l'instant de la dernière requête traitée, il est strictement inférieur à celui de la première requête à traiter :

$$r_1 - l_1 \neq \emptyset \Rightarrow \ c_1 < \mathsf{min}(\mathsf{dom}(r_1 - l_1))$$

Enfin, le nombre de requêtes en attente est borné par un nombre entier NN que l'on pourra préciser plus tard. On introduit le nouvel événement : *ctr_cmd* qui indique que le contrôleur envoie une requête aux disques. La requête traitée par l'ancien événement *dsk_rec* est celle

dont le temps d'attente est minimal dans l'ensemble des requêtes en attente. Le nouveau système abstrait devient :

```
SYSTEM
    SCSI2_1  REFINES  SCSI2_0
VARIABLES
    l_1, c_1, r_1
INVARIANT
    l_1 = l_0 ∧ c_1 = c_0 ∧      /* invariant de liaison */
    r_1 ∈ ℕ ⇸ REQ ∧
    dom(r_1) ∈ 𝔽(ℕ) ∧
    l_1 ⊆ r_1 ∧
    r_1 − l_1 ≠ ∅ ⇒ c_1 < min(dom(r_1 − l_1))
INITIALISATION
    l_1, c_1, r_1 := ∅, 0, ∅
EVENTS
    ctr_cmd =                    /* envoi d'une commande */
        ANY t, y WHERE
            t ∈ ℕ − dom(r_1 − l_1) ∧ c_1 < t ∧
            y ∈ REQ ∧ card(r_1 − l_1) < NN
        THEN
            r_1 := r_1 ∪ {t ↦ y}
        END
    ;
    dsk_rec =                    /* traitement d'une requête */
        SELECT
            r_1 − l_1 ≠ ∅
        THEN
            LET t BE
                t = min(dom(r_1 − l_1))
            IN
                c_1 := t ||
                l_1 := l_1 ∪ {t ↦ r_1(t)}
            END
        END
END
```

La méthode B permet d'indiquer une condition de vivacité qui exprime que n'importe quelle requête x en attente d'être traitée sera bien effectivement traitée.

1.1.3 Deuxième raffinement: introduction des tampons abstraits

Le dernier raffinement de cette première étape consiste à ne garder que les requêtes en attente. Le changement de variables fait introduire w_2 qui est l'ensemble des requêtes en attente. Pour pouvoir contrôler le fait qu'elles sont consommées, on leur attache non plus le

moment où elles seront satisfaites, mais le temps d'attente maximum supposé. Comme ces temps d'attente sont distincts (on ne peut pas satisfaire plusieurs requêtes au même moment [1]) w_2 est une fonction injective. Les événements sont modifiés en conséquence du changement de variables :

```
SYSTEM
    SCSI2_2  REFINES  SCSI2_1
VARIABLES
    w_2
INVARIANT
    w_2 ∈ REQ ⤔ ℕ_1  ∧
    w_2 = (r_1 − l_1)^−1 ; minus(c_1)  ∧  card(w_2) ≤ NN
INITIALISATION
    w_2 := ∅
EVENTS
    ctr_cmd =                    /* envoi d'une commande */
        ANY y, w WHERE
            w ∈ ℕ_1 − ran(w_2)  ∧  y ∈ REQ  ∧  card(w_2) < NN
        THEN
            w_2 := w_2 ∪ {y ↦ w}
        END
    ;
    dsk_rec =                    /* traitement d'une requête */
        SELECT
            w_2 ≠ ∅
        THEN
            LET t BE
                t = min(ran(w_2))
            IN
                w_2 := (w_2 ⩥ {t}) ; minus(t)
            END
        END
END
```

L'opération minus est de type $\mathsf{minus} \in \mathbb{N}_1 \nrightarrow (\mathbb{N}_1 \rightarrow \mathbb{N}_1)$ et elle n'est définie que pour des entiers strictement plus grands que le paramètre qui suit. L'effet est : $\mathsf{minus}(n)(m) = m - n$.

1.2 Deuxième étape : gestion des files

1.2.1 Exemple avec un seul disque

La description du système abstrait du paragraphe 1.1.3 comporte un événement d'envoi de requêtes et une politique générale de satisfaction de ces requêtes. Le problème consiste maintenant à dériver le système SCSI-2 simplifié à partir de cette dernière spécification.

1. On prend cette hypothèse, puisque c'est l'accès au bus qui indique le moment où la requête est satisfaite.

Voyons d'abord comment cela se ferait s'il n'y avait qu'un seul disque. On a alors :

$$NN = maxi$$

où maxi est la taille d'une file (8 dans l'étude de cas). La file est modélisée par une séquence de requêtes. L'ordre de "consommation" des requêtes est donné par l'ordre dans la file, c'est-à-dire, en B, par l'indice des éléments de la séquence (ou encore le domaine de la séquence, vue comme une fonction d'un intervalle d'entiers dans les éléments). Le raffinement conduit à une description simple, due aux primitives du langage B :

```
SYSTEM                          /* cas particulier d'un seul disque */
    SCSI2_3  REFINES  SCSI2_2
VARIABLES
    q_3
INVARIANT
    q_3 ∈ seq(REQ)  ∧
    q_3 = w_2^-1  ∧  size(q_3) ≤ maxi
INITIALISATION
    q_3 := []
EVENTS
    ctr_cmd =                   /* envoi d'une commande */
        ANY y WHERE
            y ∈ REQ  ∧  size(q_3) < maxi
        THEN
            q_3 := q_3 <- y
        END
    ;
    dsk_rec =                   /* traitement d'une requête */
        SELECT
            q_3 ≠ []
        THEN
            q_3 := tail(q_3)
        END
END
```

Le point important de cette spécification est que l'on est capable d'exprimer et de *démontrer* que toutes les requêtes sont satisfaites. Pour cela, il faut décrire une "*modalité*" qui est une formule de logique temporelle disant que, à partir de tout état tel que la requête x est dans la file q_3, alors on arrive inévitablement dans un état tel que x n'est plus dans la file (et donc a été satisfaite). C'est un prédicat de la forme :

$$\Box(x\ dans\ q_3 \Rightarrow \Diamond(x\ n'est\ plus\ dans\ q_3))$$

En B, on doit également donner un invariant et un variant pour pouvoir démontrer une telle propriété (ici, l'invariant est inutile) :

ANY x WHERE
$x \in \mathsf{ran}(q_3)$
LEADSTO
$x \notin \mathsf{ran}(q_3)$
VARIANT
$maxi + 1 - \mathsf{size}(q_3) + 2 * (q_3^{-1}(x) - 1)$
END

La preuve de cette formule se fait en montrant que pour chaque événement $\mathcal{E}$, le variant V décroît strictement :

$$I \Rightarrow [n := V][\mathcal{E}](V < n)$$

Si on pose : $m = \mathsf{size}(q_3)$ et $k = q_3^{-1}(x)$, alors on a :

$$\begin{array}{lcllcl} \mathsf{size}(q_3 \leftarrow y) & = & m+1 & \mathsf{size}(\mathsf{tail}(q_3)) & = & m-1 \\ (q_3 \leftarrow y)^{-1}(x) & = & k & (\mathsf{tail}(q_3))^{-1}(x) & = & k-1 \end{array}$$

Pour les deux événements, cela donne les inégalités :

$$\begin{array}{rcl} maxi + 1 - (m+1) + 2 * (k-1) & < & maxi + 1 - m + 2 * (k-1) \\ maxi + 1 - (m-1) + 2 * (k-1-1) & < & maxi + 1 - m + 2 * (k-1) \end{array}$$

1.2.2 Modélisation avec plusieurs disques

On introduit les constantes du système SCSI-2. Le système est composé d'un contrôleur CTR et de disques DSK. Ces éléments (ou "devices" DEV) sont identifiés par des entiers. D'autre part, les tampons attachés aux disques sont des files d'attente, modélisées par des séquences de requêtes. Dans le cas d'une seule file, le "temps d'attente" des requêtes était modélisé par l'indice des éléments dans la file (cf. l'invariant de $SCSI2_3$). Ce ne peut plus être le cas ici, puisque les indices d'éléments de files différentes peuvent être égaux. On va donc garder, pour chaque requête le temps abstrait indiqué par w_2, quitte probablement à le supprimer plus tard. On dispose donc de deux informations : les éléments qui sont dans les files d'attente des disques et les temps estimés pour chacun de ces éléments. Le système dépend de deux paramètres nn et $maxi$ Le nombre nn, entier non nul, est le nombre de disques et $maxi$ est la taille maximum des files. Par rapport aux constantes déjà introduites, on : $nn * maxi = NN$. Dans la version standard, les disques sont numérotés de 0 à $nn - 1$ et le contrôleur a le numéro nn (ces arrangements peuvent être modifiés pour tester des configurations particulières). D'où les déclarations mathématiques de ces objets :

$$\begin{array}{ll} nn \in \mathsf{NAT}_1 & DSK = 0..nn-1 \\ maxi \in \mathsf{NAT}_1 & DEV = 0..nn \\ CTR = nn & BUF = 1..maxi \end{array}$$

L'état (abstrait) du problème consiste à savoir quelles sont les requêtes en attente pour chaque disque et leur temps d'attente estimé.

$$
\begin{array}{l}
buf_4 \ \in \ DSK \rightarrow \mathsf{seq}(REQ) \\
att_4 \ \in \ DSK \rightarrow \mathsf{seq}(\mathbb{N}_1) \\
buf_4 \ ; \mathsf{size} = att_4 \ ; \mathsf{size}
\end{array}
$$

La dernière ligne indique que les requêtes et les attentes sont des séquences de même longueur. L'opérateur $\otimes$ sur des séquences de même longueur produit la séquence des couples d'éléments. L'invariant de liaison dit simplement que l'union des couples de toutes les files d'attente des disques est égale à w_2 :

$$
w_2 = \bigcup dsk \cdot (dsk \in DSK \mid \mathsf{ran}(buf_4(dsk) \otimes att_4(dsk)))
$$

L'événement *ctr_cmd* "choisit" un disque dsk pour traiter les requêtes de manière non-déterministe. En revanche, *dsk_rec* doit prendre le disque pour lequel la requête en tête de file a un temps d'attente minimum, d'après les spécifications du paragraphe 1.1.3. On voit donc que la politique d'arbitrage (le fait de distinguer un disque prioritaire, etc.) doit être reflétée dans les valeurs de att_4 ou dans des valeurs cachées dans les structures, comme c'est le cas s'il n'y a qu'un seul disque. La structure de données impose automatiquement des contraintes sur le choix des poids w attribués aux requêtes. Ces contraintes doivent faire partie de l'invariant du système. Une première contrainte est que l'ordonnancement des requêtes dans les files soit cohérent avec les valeurs d'attente (condition *ordo_file*) :

$$
\begin{array}{lll}
ordo\_file & \Leftrightarrow & \forall\, dsk \cdot (dsk \in DSK \Rightarrow \\
& & \quad (\forall\, i, j \cdot (i \in 1..maxi \ \wedge \ j \in 1..maxi \Rightarrow \\
& & \quad\quad (i < \mathsf{size}(att_4(dsk)) \ \wedge \ j < \mathsf{size}(att_4(dsk)) \ \wedge \ i < j \Rightarrow \\
& & \quad\quad\quad att_4(dsk)(i) < att_4(dsk)(j)))))
\end{array}
$$

Il faut d'autre part que les valeurs d'attente d'un disque moins prioritaire soient plus grandes que les valeurs d'attente d'un disque plus prioritaire :

$$
\begin{array}{lll}
ordo\_prio & \Leftrightarrow & \forall\, dsk, dsk' \cdot (dsk \in DSK \ \wedge \ dsk' \in DSK \ \wedge \ dsk < dsk' \Rightarrow \\
& & \quad (att_4(dsk) \neq [\,] \ \wedge \ att_4(dsk') \neq [\,] \Rightarrow \\
& & \quad\quad \mathsf{min}(\mathsf{ran}(att_4(dsk))) > \mathsf{max}(\mathsf{ran}(att_4(dsk')))))
\end{array}
$$

Les conditions ci-dessus imposent un renforcement des gardes au moment des choix de w en fonction d'un disque dsk, pour l'événement *ctr_cmd*.

1. Il faut que le poids associé à une requête d'un disque dsk soit plus grand que le temps d'attente du dernier élément de la file :

$$
ordo\_dsk\_file(dsk, w) \ \Leftrightarrow \ (dsk \neq [\,] \Rightarrow \ w > \mathsf{last}(att_4(dsk)))
$$

2. Lorsqu'une requête est envoyée à un disque, sa valeur d'attente doit être inférieure aux valeurs d'attente de toutes les requêtes des disques moins prioritaires, donc de la première valeur de leur file :

$$ordo\_dsk\_inf(dsk, w) \quad \Leftrightarrow$$
$$\forall\, dsk' \cdot (dsk' \in DSK \ \wedge\ att_4(dsk') \neq [\,] \ \wedge\ dsk' < dsk \Rightarrow\ w < \mathsf{first}(att_4(dsk')))$$

3. Lorsqu'une requête est envoyée à un disque, sa valeur d'attente doit être supérieure aux valeurs d'attente de toutes les requêtes des disques plus prioritaires, donc de la dernière valeur de leur file :

$$ordo\_dsk\_sup(dsk, w) \quad \Leftrightarrow$$
$$\forall\, dsk' \cdot (dsk' \in DSK \ \wedge\ att_4(dsk') \neq [\,] \ \wedge\ dsk < dsk' \Rightarrow\ w > \mathsf{last}(att_4(dsk')))$$

En conséquence de toutes ces définitions, un disque dsk, dont la file est non vide, est *prioritaire* ssi l'élément de tête de la file a un temps d'attente inférieur à tous les autres éléments de tête de file des autres disques (dont la file n'est pas vide).

$$\begin{aligned} prioritaire(dsk) \quad \Leftrightarrow \quad & att_4(dsk) \neq [\,] \ \wedge \\ & \forall\, dsk' \cdot (dsk' \in DSK \ \wedge\ dsk' \neq dsk \ \wedge\ att_4(dsk') \neq [\,] \Rightarrow \\ & \qquad \mathsf{first}(att_4(dsk)) < \mathsf{first}(att_4(dsk'))) \end{aligned}$$

Enfin, les conditions dans lesquelles les événements ctr_cmd et dsk_rec peuvent se produire sont évidentes et impliquent les conditions abstraites : (1) on ne peut envoyer un message que si la file du disque en question n'est pas pleine; (2) le fait d'envoyer un traitement signifie qu'il y avait des messages à traiter dans la file du disque. Ces événements modifient l'état d'une manière appropriée. La spécification théorique du raffinement de $SCSI2_2$ dans le cas où il y a plusieurs disques est :

SYSTEM
$SCSI2_4$ REFINES $SCSI2_2$
VARIABLES
buf_4, att_4
INVARIANT
$buf_4 \in DSK \rightarrow \mathsf{seq}(REQ) \ \wedge$
$att_4 \in DSK \rightarrow \mathsf{seq}(\mathbb{N}_1) \ \wedge$
$buf_4 \ ; \mathsf{size} \in DSK \rightarrow BUF$
$buf_4 \ ; \mathsf{size} = att_4 \ ; \mathsf{size} \ \wedge$
$ordo\_file \ \wedge \ ordo\_prio \ \wedge$
$w_2 = \bigcup dsk \cdot (dsk \in DSK \mid \mathsf{ran}(buf_4(dsk) \otimes att_4(dsk)))$
INITIALISATION
$buf_4, att_4 := DSK \times \{[\,]\}, DSK \times \{[\,]\}$

```
EVENTS
    ctr_cmd =                    /* envoi d'une commande au disque dsk */
        ANY dsk, y, w WHERE
            dsk ∈ DSK  ∧  size(buf4(dsk)) < maxi  ∧  y ∈ REQ  ∧  w ∈ ℕ1  ∧
            ordo_dsk_file(dsk, w)  ∧  ordo_dsk_inf(dsk, w)  ∧  ordo_dsk_sup(dsk, w)
        THEN
            buf4(dsk) := buf4(dsk) ← y ||
            att4(dsk) := att4(dsk) ← w
        END
    ;
    dsk_rec =                    /* fin de traitement d'une commande */
        ANY dsk WHERE
            prioritaire(dsk)
        THEN
            buf4(dsk) := tail(buf4(dsk)) ||
            att4 := att4 ⩤ {dsk ↦ tail(att4(dsk))} ; minus_seq(att4(dsk)(1))
            END
        END
END
```

Notons que les conditions $ordo\_dsk\_file(dsk, w) \wedge ordo\_dsk\_inf(dsk, w) \wedge ordo\_dsk\_sup(dsk, w)$ impliquent la condition abstraite : $w \in \mathbb{N}_1 - \mathsf{ran}(w_2)$. On a introduit l'opération minus_seq qui retranche la valeur en paramètre des éléments des séquences pour produire de nouvelles séquences (à la manière de minus) :

$$\mathsf{minus\_seq} \in \mathbb{N}_1 \leftrightarrow (\mathsf{seq}(\mathbb{N}_1) \rightarrow \mathsf{seq}(\mathbb{N}_1))$$

La précondition de cet opérateur est que le premier paramètre est strictement plus grand que les éléments de la séquence sur laquelle s'applique l'opération, ce qui est bien le cas dans notre spécification.

Discussion : L'analyse de cette spécification montre que si les requêtes sont envoyées sur plusieurs disques, il y a des situations dans lesquelles le contrôleur ne pourra plus envoyer de requêtes sur le disque le plus prioritaire, *même si sa file n'est pas pleine*. Il est facile de voir, en effet, que si l'on a mis une requête sur un disque qui n'est pas le plus prioritaire avec un temps d'attente w (forcément fini), alors d'une part, le nombre d'événements ctr_cmd est borné, soit par les tailles des files, soit par le fait de la condition $ordo\_dsk\_inf$. D'autre part, le nombre d'événements de dsk_rec est borné par w, puisque cette valeur décroît strictement à chaque occurrence de l'événement. La spécification est donc bien équitable. Il reste à savoir s'il est possible de raffiner cette spécification en un véritable algorithme qui prenne en compte cette équité. Si l'on est capable de prouver les raffinements successifs, alors l'algorithme sera équitable par construction. Si l'on n'en est pas capable, c'est que l'algorithme final aura perdu certaines propriétés.

> *C'est bien ce qui se passe si l'on continue le développement en essayant d'aboutir à l'algorithme qui ne tient compte que des priorités et des files.*

1.3 Troisième étape : vers une distribution de l'état

À partir de maintenant, on s'intéresse uniquement à la décomposition du problème en plusieurs processus communicants. On oublie donc toute l'information sur les attentes, pour ne travailler qu'au niveau des structures d'information données. Du coup, on perd la propriété d'équité du système, puisque la structure avec plusieurs files et un disque prioritaire est insuffisante pour l'assurer [2].

Dans cette troisième étape, on veut distinguer les informations attachées au contrôleur et celles attachées aux disques. En effet, le contrôleur ne doit envoyer de messages que s'il sait que la file du disque en question n'est pas pleine. Donc, le contrôleur doit avoir une image globale des tailles courantes des files (variable $taille_5$). D'où le nouvel état :

$$\begin{array}{l} taille_5 \ \in \ DSK \rightarrow BUF \\ buf_5 \ \in \ DSK \rightarrow \mathsf{seq}(REQ) \end{array}$$

avec la relation : $taille_5 = buf_5$; size. Le nouveau système $SCSI2_5$ gère à la fois les requêtes et les longueurs des files.

SYSTEM
  $SCSI2_5$
VARIABLES
  $buf_5, taille_5$
INVARIANT
  $buf_5 \ \in \ DSK \rightarrow \mathsf{seq}(REQ) \ \wedge$
  $taille_5 \ \in \ DSK \rightarrow BUF \ \wedge$
  $taille_5 = buf_5$; size
INITIALISATION
  $buf_5, taille_5 := DSK \times \{[]\}, DSK \times \{0\}$
EVENTS
  $ctr\_cmd =$ /* envoie d'une commande au disque dsk */
    ANY dsk, y WHERE
      $dsk \in DSK \ \wedge \ taille_5(dsk)) < maxi \ \wedge \ y \in REQ$
    THEN
      $buf_5(dsk) := buf_5(dsk) \leftarrow y \ ||$
      $taille_5(dsk) := taille_5(dsk) + 1$
    END
  ;
  $dsk\_rec =$ /* fin de traitement d'une commande */
    ANY dsk WHERE
      $dsk \in DSK \ \wedge \ prioritaire(dsk)$
    THEN
      $buf_5(dsk) := \mathsf{tail}(buf_5(dsk)) \ ||$
      $taille_5(dsk) := taille_5(dsk) - 1$
    END
END

2. Il faudrait reprendre l'étude avec décomposition et préservation de la propriété.

On a gardé le prédicat $prioritaire(dsk)$ (dont la définition sera donnée plus tard) pour le choix du disque qui doit renvoyer un acquittement. On sait simplement que si un disque est prioritaire, sa file de requêtes n'est pas vide.

1.4 Quatrième étape : introduction du bus

1.4.1 Analyse du raffinement proposé

Dans cette étape, on fait apparaître explicitement le bus entre le contrôleur et les disques. Il s'agit d'une variable abstraite (bus), mais elle est caractéristique des systèmes distribués où chaque composant ne peut communiquer que par un bus ou un canal. Le bus est vide ou bien il contient un seul message de la forme : $\{CMD \mapsto (dsk, y)\}$ ou $\{REC \mapsto (dsk, y)\}$, suivant qu'il s'agit d'une requête y envoyée au disque dsk ou d'une réception en provenance du disque dsk. Les "devices" n'accèdent au bus que s'il est vide (approximation de l'utilisation du bus). Pour les événements, on dissocie l'envoi d'un message de sa réception, ce qui ne veut pas forcément dire qu'il ne seront pas synchronisés ensuite.

ctr_cmd : le contrôleur envoie une commande sur le bus,
dsk_cmd : un disque réceptionne une commande du bus,
dsk_rec : un disque envoie une fin de traitement sur le bus,
ctr_rec : le contrôleur réceptionne une fin de traitement.

L'état est plus complexe, puisque les variables attachées aux disques et celle attachée au contrôleur peuvent avoir des informations légèrement différentes. On introduit les informations qui transitent sur le bus. L'ensemble des commandes est :

$$ACT = \{CMD, REC\}$$

Le bus est caractérisé par l'invariant :

$$bus \in ACT \leftrightarrow (DSK \times REQ) \ \wedge \\ (\mathsf{card}(bus) = 0 \ \vee \ \mathsf{card}(bus) = 1)$$

On note $taille_6$ la variable côté contrôleur et buf_6 la variable côté disques, avec les typages habituels. L'invariant de liaison, qui relie $taille_5$ et buf_5 avec bus, $taille_6$ et buf_6, est exprimé par :

$$(bus = \emptyset \ \Rightarrow \ (taille_6 = taille_5 \ \wedge \ buf_6 = buf_5)) \ \wedge \\ \forall u, y \cdot (u \in DSK \ \wedge \ y \in REQ \ \wedge \ bus = \{CMD \mapsto (u, y)\} \ \Rightarrow \\ \qquad (taille_6 = taille_5 \ \wedge \ (buf_6 \lhd\!\!+ \{u \mapsto buf_6(u) \lhd\!\!- y\}) = buf_5)) \ \wedge \\ \forall v, z \cdot (v \in DSK \ \wedge \ z \in REQ \ \wedge \ bus = \{REC \mapsto (v, z)\} \ \Rightarrow \\ \qquad ((taille_6 \lhd\!\!+ \{v \mapsto taille_6(v) - 1\}) = taille_5 \ \wedge \ buf_6 = buf_5))$$

1.4.2 Description des événements

Les événements ne présentent pas de difficultés particulières (la clause d'initialisation est omise) :

EVENTS

$ctr\_cmd =$ /* CTR envoie une commande sur le bus */

ANY dsk, y WHERE

$dsk \in DSK \ \wedge \ y \in REQ \ \wedge \ taille_6(dsk) < maxi \ \wedge \ bus = \emptyset$

THEN

$bus := \{CMD \mapsto (dsk, y)\} \ ||$

$taille_6(dsk) := taille_6(dsk) + 1$

END

;

$dsk\_cmd =$ /* DSK réceptionne une commande */

ANY dsk, y WHERE

$dsk \in DSK \ \wedge \ y \in REQ \ \wedge \ bus = \{CMD \mapsto (dsk, y)\}$

THEN

$bus := \emptyset \ ||$

$buf_6(dsk) := buf_6(dsk) \leftarrow y$

END

;

$dsk\_rec =$ /* DSK envoie une commande sur le bus */

ANY dsk WHERE

$dsk \in DSK \ \wedge \ prioritaire(dsk) \ \wedge \ bus = \emptyset$

THEN

$bus := \{REC \mapsto (dsk, \mathsf{first}(buf_6(dsk)))\} \ ||$

$buf_6(dsk) := \mathsf{tail}(buf_6(dsk))$

END

;

$ctr\_rec =$ /* CTR réceptionne et fin de traitement */

ANY dsk, y WHERE

$dsk \in DSK \ \wedge \ y \in REQ \ \wedge \ bus = \{REC \mapsto (dsk, y)\}$

THEN

$bus := \emptyset \ ||$

$taille_6(dsk) := taille_6(dsk) - 1$

END

END

Comme il y a de nouveaux événements et que les gardes des événements "anciens" sont différentes (introduction de la condition $bus = \emptyset$), il faut s'assurer dans les vérifications qu'il y a des événements possibles pour le cas $bus \neq \emptyset$ et d'autre part que les nouveaux événements ne peuvent pas empêcher les anciens événements de se produire. Cette dernière condition devraient être exprimée par un variant.

1.5 Cinquième étape : la politique d'arbitrage

1.5.1 Analyse générale

Dans la cinquième étape, on introduit la politique d'arbitrage. Cette politique est définie dans le document en plusieurs endroits. On peut trouver les indications suivantes :

> *1- Cette politique d'arbitrage est basée sur les priorités : si plusieurs périphériques veulent simultanément accéder au bus, c'est le périphérique dont le numéro SCSI est le plus grand qui l'emporte.*
>
> *2- Il n'y a pas d'arbitre centralisé pour distribuer les autorisations d'accès au bus. Chaque périphérique doit surveiller les demandes émanant des autres périphériques et n'accéder au bus que si aucun autre périphérique plus prioritaire n'a effectué de demande en même temps.*

D'autre part, on a des informations de mise en œuvre de cette politique, du point de vue de l'implantation :

> *3- Physiquement, l'arbitrage est implémenté par huit fils, dont chaque périphérique peut consulter la valeur. Tout périphérique de numéro n désirant accéder au bus doit porter le $n^{ième}$ fil au niveau haut.*
>
> *4-Il s'agit d'une programmation par filtrage : chaque périphérique pose ses propres contraintes pour accéder au bus, accès qui aura lieu si les contraintes posées par les périphériques sont satisfaites.*
>
> *5- On doit caractériser le fait qu'un périphérique ne désire pas accéder au bus.*
>
> *6- On doit caractériser la situation dans laquelle le $n^{ième}$ périphérique indique qu'il désire accéder au bus et constate (en examinant les demandes des autres périphériques) qu'il y est autorisé.*

En B, les huit fils sont modélisés par un ensemble qui contient les demandes d'accès. Un fil i au niveau haut est équivalent au fait que le périphérique numéro i est dans l'ensemble. D'où la variable $wire$:

$$wire \in \mathbb{P}(DEV)$$

Un périphérique kk est autorisé à utiliser le bus si d'une part, le bus est libre et, si d'autre part, il est prioritaire sur les autres. Cela se traduit par :

$$autorise(kk) \Leftrightarrow bus = \emptyset \wedge wire \neq \emptyset \wedge kk = \mathsf{max}(wire)$$

On introduit les nouveaux événements de demande d'accès au bus :

ctr_acc :	le contrôleur demande l'accès au bus
dsk_acc :	un disque demande l'accès au bus

On pose un certain nombre de conditions qui limitent les occurrences de ces demandes ou relachement d'accès. Dans la description fournie [GM98], les demandes sont faites simultanément par le rendez-vous `ARB` et de manière fugitive aux moments suivants :

1. Pour le contrôleur, lorsqu'il souhaite engager un envoi de commande à un disque dsk, sous la condition que le tampon du disque en question ne soit pas plein (d'après la connaissance qu'il en a) : $taille_7(dsk) < maxi$.

2. Pour les disques dsk, dès qu'ils ont des messages à traiter : $buf_7(dsk) > 0$, ils doivent faire des demandes permanentes jusqu'à ce qu'ils deviennent prioritaires pour renvoyer leur acquittement de traitement. En B, cela veut dire qu'ils sont dans l'ensemble $wire$.

3. Suite à la remarque précédente, un disque dsk ne peut relacher le bus que si son tampon est vide : $buf_7(dsk) = 0$.

1.5.2 Problèmes posés par la modélisation par événements

On cherche à faire une modélisation par événements. La seule façon de lier des événements est de renforcer les gardes pour assurer, par exemple, qu'un événement a et toujours suivi par un événement b. Dans notre modélisation, on choisit de séparer la demande d'arbitrage du fait que la communication est possible. De plus, la communication fait intervenir deux événements distincts, à savoir la mise sur le bus et la récupération de l'information du bus. Tout cela a un effet dans l'ordonancement des événements. Dans la description Lotos, il y a le cas :

Il demande le bus et ne l'obtient pas.

sur lequel le système sort simplement de la consultation des demandes et ne fait rien. En B, cette situation n'a pas à être modélisée spécifiquement, puisque si une garde n'est pas vraie, l'événement en question ne peut pas se produire. Si l'on regarde la description Lotos, on a les suites obligées d'événements (mais peut-être entrelacées avec d'autres) :

$ctr\_acc \rightsquigarrow ctr\_cmd \rightsquigarrow dsk\_cmd$	envoi de CMD
$dsk\_acc \rightsquigarrow dsk\_cmd \rightsquigarrow dsk\_rec \rightsquigarrow ctr\_rec$	traitement disque

En particulier, si le contrôleur est engagé dans une transaction avec un disque (non plein), cette information doit être préservée tant que le contrôleur n'a pas déposé son message sur le bus. On traduit cela en B en utilisant deux nouvelles variables $dskrq$ (disque requis) et msg (message / requête envoyé).

$$dskrq \in DEV$$
$$msg \in REQ$$

La variable $dskrq$ vaut dd si le contrôleur est engage avec le disque dd et elle vaut CTR si le contrôleur n'est pas engagé. De plus, la variable msg n'est significative que lorsque le contrôleur est engagé avec un disque. D'où la définition de prédicat :

$$ctr\_engage \Leftrightarrow dskrq \in DSK$$

Lorsque le contrôleur est ainsi engagé dans une communication, il ne peut pas se re-engager (perte possible d'information), ni relacher le bus. Enfin, lorsque le contrôleur est engagé avec un disque dd, il a la connaissance que celui-ci n'est pas plein :

$$ctr\_engage \Rightarrow taille_7(dskrq) < maxi$$

1.5.3 Description des événements de cette phase

On est prêt maintenant à décrire les événements en B. On peut remarquer qu'il y a deux cas à envisager pour dsk_rec suivant que la file des requêtes devient vide ou pas. Si la file devient vide, alors le disque doit être retiré de l'ensemble qui accède au bus.

INITIALISATION
$bus, taille_7, buf_7, wire, dskrq := \emptyset, DSK \times \{0\}, DSK \times \{[]\}, \emptyset, CTR \;||$
$msg :\in REQ$
EVENTS
$ctr\_acc =$ /* CTR demande l'accès au bus */
ANY dsk, y WHERE
$dsk \in DSK \;\wedge\; y \in REQ \;\wedge\; taille_7(dsk) < maxi \;\wedge\; \neg\, ctr\_engage$
THEN
$wire := wire \cup \{CTR\} \;||$
$dskrq, msg := dsk, y$
END
;
$ctr\_cmd =$ /* CTR envoie une requête sur le bus */
SELECT
$ctr\_engage \;\wedge\; autorise(CTR)$
THEN
$bus := \{CMD \mapsto (dskrq, msg)\} \;||$
$taille_7(dskrq) := taille_7(dskrq) + 1 \;||$
$dskrq := CTR \;||$
$wire := wire - \{CTR\}$
END
;
$ctr\_rec =$ /* CTR réceptionne une fin de traitement */
ANY dsk, y WHERE
$dsk \in DSK \;\wedge\; y \in REQ \;\wedge\; bus = \{REC \mapsto (dsk, y)\}$
THEN
$bus := \emptyset \;||$
$taille_7(dsk) := taille_7(dsk) - 1$
END
;
$dsk\_acc =$ /* le disque dsk veut accéder au bus */
ANY dsk, y WHERE
$dsk \in DSK \;\wedge\; y \in REQ \;\wedge\; bus = \{CMD \mapsto (dsk, y)\} \;\wedge\; dsk \notin wire$
THEN
$wire := wire \cup \{dsk\}$
END

```
EVENTS (suite)
    ;
    dsk_cmd =                    /* le disque dsk réceptionne une requête */
        ANY dsk, y WHERE
            dsk ∈ DSK  ∧  y ∈ REQ  ∧  bus = {CMD ↦ (dsk, y)}  ∧  dsk ∈ wire
        THEN
            bus := ∅ ||
            buf7(dsk) := buf(dsk) ⇐ y
        END
    ;
    dsk_rec =                    /* le disque dsk envoie une fin de traitement */
        ANY dsk WHERE
            dsk ∈ DSK  ∧  buf7(dsk) > 0  ∧  autorise(dsk)
        THEN
            bus := {REC ↦ (dsk, first(buf7(dsk)))} ||
            buf7(dsk) := tail(buf7(dsk));
            IF buf7(dsk) = [] THEN
                wire := wire − {dsk}
            END
        END
END
```

On remarque que dans cette version le prédicat *prioritaire* se réduit au fait que la file d'attente n'est pas vide et que le disque a le numéro maximum dans l'ensemble *wire* :

$$prioritaire(dsk) \;\Leftrightarrow\; buf_7(dsk) > 0 \;\wedge\; dsk \in wire \;\wedge\; dsk = \mathsf{max}(wire)$$

2 Validation B du système SCSI-2

Il reste à valider les spécification avec l'outil B. *La première partie n'a pas été validée. La deuxième partie l'a été avec une version qui ne faisait pas intervenir les requêtes y (donc une version plus proche de la solution Lotos).*

Les validations comprennent les preuves de raffinement, les conditions de non-blocage et les conditions de non-limitation des occurrences d'événements (ces deux dernières conditions devant être engendrées à la main sous forme d'assertions).

Une suite du travail consisterait à voir comment vérifier les propriétés de sûreté et de vivacité données dans le cahier des charges et à développer jusqu'au bout une solution équitable de l'algorithme avec priorités.

Enfin, on pourrait étudier comment la spécification B *peut être transformée pour construire la solution Lotos, ou tout autre solution dans un langage quelconque (est-ce possible? est-ce utile? etc.)*

Références

[Abr96] Jean-Raymond Abrial. Extending B without changing it (for developing distributed systems). In H. Habrias, editor, *Proc. of the 1st B Conference, Nantes*, pages 169–190. IRIN, ISBN: 2-906082-25-2, 1996.

[Abr98] Jean-Raymond Abrial. On B. In D. Bert, editor, *B'98: Recent Advances in the Development and Use of the B Method, Proc. of the 2nd Int. B Conference, Montpellier*, pages 1–8. Springer, 1998.

[AM98] Jean-Raymond Abrial and Louis Mussat. Introducing dynamic constraints in B. In D. Bert, editor, *B'98: Recent Advances in the Development and Use of the B Method, Proc. of the 2nd Int. B Conference, Montpellier*, pages 83–128. Springer, 1998.

[GM98] Hubert Garavel and Radu Mateescu. Modélisation d'un système d'entrées-sorties SCSI-2, Version 3.0. Technical report, Action VERDON, INRIA Rhône-Alpes, octobre 1998.