Simulation interactive du modèle de gestion d'interconnexion de commutateurs à l'aide de LOBSTERS et MODE

Olivier Festor, Emmanuel Nataf, Laurent Andrey

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Simulation interactive du modèle de gestion d'interconnexion de commutateurs à l'aide de LOBSTERS et MODE

Olivier Festor, Emmanuel Nataf, Laurent Andrey

N° 2790

Février 1996

PROGRAMME 1

ISSN 0249-6399

Simulation interactive du modèle de gestion d'interconnexion de commutateurs à l'aide de LOBSTERS et MODE

Olivier Festor, Emmanuel Nataf, Laurent Andrey

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet RESEDAS

Rapport de recherche n° 2790 — Février 1996 — 31 pages

Résumé :

En raison de la complexité croissante des modèles de l'information basés sur l'approche OSI, les concepteurs de catalogues d'objets gérés expriment des besoins de mise en œuvre de méthodes formelles pour valider les aspects liés au comportement de leurs spécifications. Dans le cadre d'une étude sur l'adéquation des méthodes formelles et outils existants pour la spécification des systèmes aux besoins des modèles de gestion de réseaux OSI, nous étudions plusieurs Techniques de Description Formelles et les outils disponibles.

Ce rapport présente l'utilisation du langage LOBSTERS ainsi que les outils MODE et CRUSADE pour la validation par simulation interactive du modèle de gestion de la configuration de l'interconnexion de commutateurs. Cette approche propose une alternative à une étude sur l'utilisation du logiciel MEC que nous avions faite sur le même modèle.

Mots-clé : CRS, Gestion de réseaux, LOBSTERS, OSI, Simulation

(Abstract: pto)

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

Interactive Simulation of the Switch Interconnection Management Model with LOBSTERS and MODE

Abstract:

Due to the growing complexity of OSI-based Information Models, designers have an increasing requirement towards formal methods in order to validate the behavioral aspects of their specification. Within a study on the adequacy of formal methods and supporting tools to the requirements for management information modeling, we investigate several Formal Description Techniques and their software environment.

This report presents the use of the LOBSTERS language combined with the MODE and CRUSADE tools for validating through simulation the behavioral dependencies between Managed Objects representing switch interconnection ressources. This investigation provides an alternative approach to a previous study on the use of the MEC software that we already applied to the same information model.

Key-words: CRS, Network Management, LOBSTERS, OSI, Simulation

1 Introduction

En administration de réseaux, la spécification des objets gérés fournit un modèle décrivant les interfaces de gestion des ressources à gérer. Cette spécification sert de base aux différentes implémentations du modèle et fait office de référence pour la conformité de ces dernières. Elle se doit donc d'être précise, non-ambigüe et valide. Si ces critères sont faciles à garantir pour des modèles simples ne comportant que peu d'objets et dont les liens sont faibles voire inexistants, il est beaucoup plus difficile de les respecter sur des modèles complexes. Or, la croissante utilisation de l'approche OSI pour la modélisation des ressources gérées entraîne tout naturellement la complexification des modèles. Ceci est principalement vrai dans le cas des aspects liés au comportement des objets qui les composent ainsi que sur les dépendances entre ces derniers.

À partir de ce moment, les notations semi-formelles telles que GDMO [1] [ISO-10165.5 92] ne permettent plus de respecter les critères précédemment identifiés et le recours à des méthodes de description formelle (FDT) se révèle opportun. À ce jour, aucune méthode de description formelle n'a été retenue officiellement au sein de l'ISO ou de l'ITU-T pour la spécification des modèles de l'information et libre choix est laissé aux concepteurs sur l'approche à utiliser.

En lien avec nos travaux sur l'expression des dépendances comportementales entre les objets gérés, nous effectuons une série d'études sur l'adéquation des approches formelles existantes aux besoins en administration de réseaux. Après une première investigation [Nataf 96] sur l'utilisation des systèmes de transitions étiquetées et du logiciel MEC [Arnold 89] développé au LABRI, nous présentons l'utilisation du langage LOBSTERS [Festor 94b, Festor 94c, Festor 94a] et du simulateur CRUSADE [Eschebach 91, Schneider 92, Orain 93] de l'environnement MODE [Festor 95] pour la spécification et la simulation interactive du modèle de la gestion de la configuration de l'interconnexion entre commutateurs [NMF 94], édité par le Network Management Forum. Ce catalogue d'objets est le même que celui utilisé dans notre précédente étude.

La démarche entreprise dans cette étude consiste à respécifier le modèle à l'aide de systèmes de règles communicants. Ceci nous permet d'aborder le modèle différemment de l'approche précédente pour les raisons suivantes:

1. spécification partielle possible (on ne décrit plus en un bloc l'ensemble du système mais chaque objet indépendamment des autres);
2. approche des dépendances différente (les propagations induites par les dépendances sont décrites par des échanges de messages explicites et non plus comme une mise à jour atomique de l'ensemble des objets participants).

De plus les outils autour de LOBSTERS nous permettent de simuler de manière interactive une configuration donnée ce qui diffère de l'approche `MEC` dans laquelle on pouvait valider des propriétés. Cela nous permettra de mieux cerner les intérêts de simuler des Bases d'Information de Gestion, d'identifier les limites de cette approche et de formuler de nouvelles exigences envers le formalisme et les outils disponibles et/ou requis.

Ce rapport est organisé comme suit. La section 2 résume les principales caractéristiques du modèle à valider. La section 3 détaille les spécifications du système à l'aide de systèmes de règles. La section 4 résume les résultats obtenus. Une conclusion sur ce travail est donnée en 5. Le lecteur trouvera en annexes la spécification complète ainsi que des exemples de l'interface CRUSADE.

2 Les caractéristiques du modèle d'interconnexion de commutateurs

Dans notre première étude sur l'utilisation du logiciel MEC pour la validation du modèle de gestion de la configuration d'interconnexion de commutateurs, le modèle est présenté de manière très détaillée. En conséquence nous n'en résumons ici que les principales caratéristiques. Pour plus de détails, le lecteur pourra consulter l'étude initiale [Nataf 96] ou le catalogue [NMF 94].

La figure 1 illustre les différents niveaux d'abstraction du modèle de gestion utilisé. Ces niveaux sont celui des systèmes réels et leurs interconnexion; une modélisation des interconnexions et finalement la définition des objets gérés associés et leur organisation au sein d'une Base d'Information de Gestion.

Le modèle (qui est un ensemble de classes et les dépendances entre elles) spécifie les composants identifiés dans l'interconnexion de commutateurs (partie *(a)* de la figure 1). Ces composants sont au niveau le plus bas, les commutateurs eux-mêmes et des liens physiques et logiques qui les interconnectent.

1. GDMO : Guideline for the Definition of Managed Object

Une première abstraction du modèle est donnée dans la partie *(b)* de la figure 1. On décompose ici le réseau (`network`) en éléments gérés (`ManagedElement`) qui sont composés d'équipements (`Equipment`). Les liens d'interconnexion entre les équipements sont décrits au travers de circuits (`circuit`) et groupes de circuits (`circuitSubGroup`). Les circuits, tout comme les groupes de circuits, disposent de points de terminaison (`circuitXtp, circuitSubgroupTerminationPoint`) rattachés à des points de terminaison physique (`trailTerminationPointBidirectional`) sur les équipements et les éléments gérés

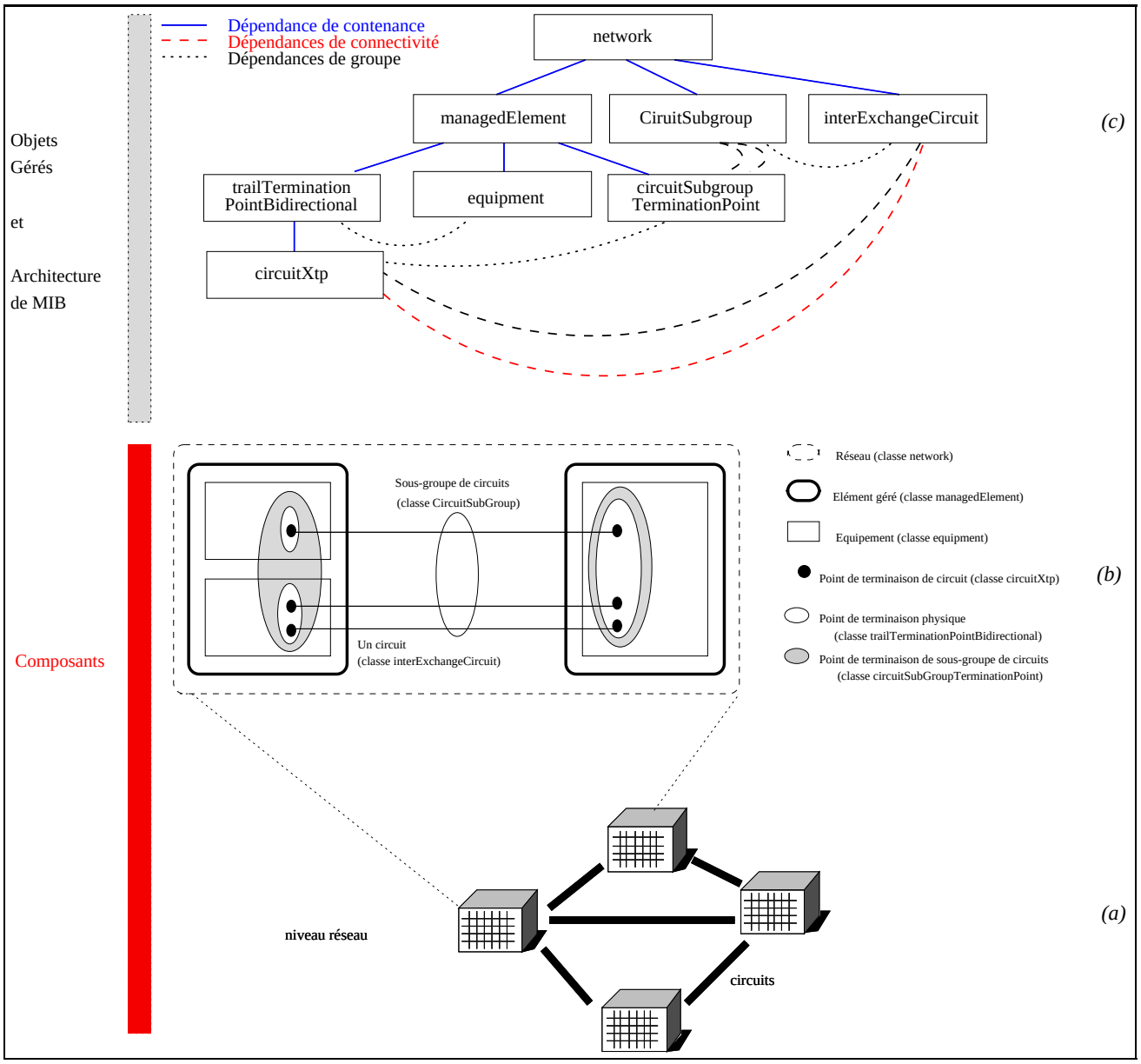


FIG. 1 – *Les niveaux d'abstraction du modèle*

Cette abstraction est ensuite spécifiée sous forme de classes d'objets au sein d'une Base d'Information de Gestion (MIB) dont l'architecture est donnée dans la partie *(c)* de la figure 1. À chaque élément du modèle (`b`), correspond une classe d'objet. Le seul lien explicite au sein d'une MIB est un lien de contenance (appelé `NAME-BINDING`) qui va servir à l'identification des instances dans la MIB. Celui-ci est utilisé dans le modèle pour définir les dépendances entre le réseau et l'ensemble des autres éléments du modèle; les éléments gérés et les équipements qui les composent. L'ensemble des ressources de connectivité (circuits et sous-groupes de circuits) sont contenus dans le réseau et n'appartiennent donc à aucun équipement. Les points de terminaison sont tous rattachés par contenance à un élément géré. Seuls les points de terminaison de circuit sont contenus dans des points de terminaison physique, eux-même contenus dans un élément géré. Sur notre schéma (figure 1

(c), le lien de contenance est exprimé en trait plein. Il existe d'autres dépendances identifiées au sein de la MIB. Ces dépendances sont de type *connectivité* (un point de terminaison de circuit termine un circuit) et de type *association* ou *groupe* (un circuit peut être associé à un sous-groupe de circuit). Ces dépendances sont réalisées dans la MIB sous forme d'attributs pointeurs [2]. Les dépendances de connectivité entraînent des comportements sur les participants. Ce sont ces comportements qui sont spécifiés de manière formelle dans notre modèle.

3 La spécification LOBSTERS

LOBSTERS [Festor 93, Festor 94b, Festor 94a] permet d'étendre le langage de spécification d'objets gérés GDMO [CCITT.X.722 92] avec une description formelle des comportements. Cette spécification formelle est réalisée sous forme de règles de comportement et d'interactions entre les objets au travers de interfaces de communication. Cette spécification est alors transcrite en langage CRS (Communicating Rule Systems) [Mackert 87, Schneider 92] c'est à dire en systèmes de règles interconnectés au travers de interfaces de communication. Cette spécification CRS peut être utilisée par différents outils et notamment par le simulateur CARUSSIM [Eschebach 91, Frot 93, Orain 93] qui facilite la simulation interactive d'une configuration de systèmes. C'est cet outil que nous avons utilisé pour simuler le modèle.

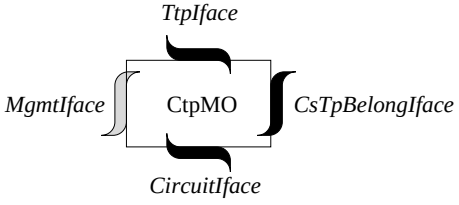
3.1 Les objets de base et les interfaces

Aux objets gérés du modèle pour la gestion de la configuration de l'interconnexion de commutateurs, nous avons associé une spécification LOBSTERS (objets + comportement + interfaces). On retrouve donc dans notre spécification les objets qui modélisent les équipements, les points de terminaison de circuit et de sous-groupe de circuits ainsi que les points de terminaison physique. Nous avons de plus, modélisé un objet opérateur qui représente l'interface de gestion capable d'émettre des opérations de gestion vers les objets. Le sous-ensemble d'objets retenu dans notre spécification formelle correspond aux objets initiallement retenus pour la modélisation en MEC.

La première étape de la spécification a consisté à définir des interfaces de communication et, pour chaque objet, son comportement ainsi que les interfaces auxquelles il a accès. Nous détaillons ici la structure de ces objets LOBSTERS.

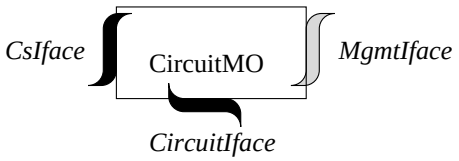
Tout objet géré dispose d'une interface de gestion (*en gris clair sur les schémas*). Les objets impliqués dans une relation peuvent disposer de plus d'une interface de communication par relation (*interfaces en noir sur les schémas*). C'est le cas de tous les objets du modèle sauf l'opérateur. Les liens représentent la mise en correspondance des interfaces de relation entre les objets du modèle.

L'objet de point de terminaison de circuit (`CtpMO`). Celui-ci dispose d'une interface de gestion (type `MgmtIface` [a]) et de trois interfaces liées aux relations dans lesquelles il peut participer. Ces relations sont la relation de terminaison de circuit (type `CircuitIface`), celle d'appartenance à un groupe de points de terminaison de circuit (type `CsTpBelongIface`) et celle d'attachement à un point de terminaison physique (type `TtpIface`).



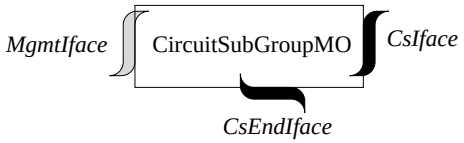
[a] Les interfaces sur les schémas des objets identifient un type d'interface. Les instances d'interfaces n'apparaîssent qu'au niveau de l'instanciation des objets eux-mêmes.

L'objet géré circuit (`CircuitMO`) possède, en plus d'une interface de gestion, deux interfaces liées à des relations. L'une concerne les interactions avec les points de terminaison de circuit (type `CircuitIface`), l'autre la relation d'appartenance à un groupe de circuits (type `CsIface`).

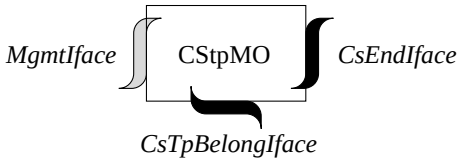


2. Cela est dû au fait qu'une instance d'objet ne peut être liée qu'à une seule autre au travers d'un lien de corrélation de noms et que ce type de relation connote toujours un lien de contenance ce qui n'est la plupart du temps pas le cas (exemple un circuit peut exister sans être rattaché à un groupe de circuits).

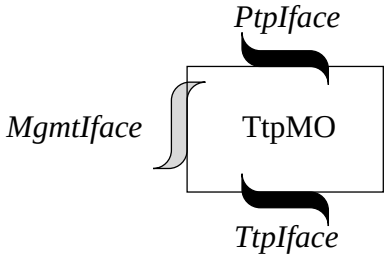
L'objet géré groupe de circuits (`CircuitSubGroupMO`) possède tout comme l'objet circuit, une interface de gestion (type `MgmtIface`) et deux interfaces pour la gestion des relations. L'une concerne les interactions avec les points de terminaison de sous-groupe de circuits (type `CsEndIface`), l'autre permet les interactions avec les circuits liés (type `CsIface`).



L'objet géré point de terminaison de sous-groupe de circuits (`CstpMO`) offre une interface de gestion (type `MgmtIface`), une interface avec le sous-groupe de circuits auquel il appartient (type `CsEndIface`) ainsi qu'une interface vers les points de terminaison de circuit qui lui sont attachés (type `CsTpBelongIface`).

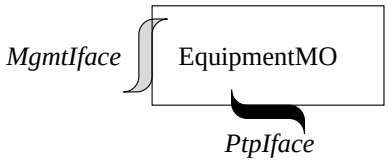


L'objet géré point de terminaison physique (`TtpMO`) dispose d'une interface de gestion (type `MgmtIface`), d'une interface de gestion de relation avec l'équipement sur lequel il se situe (type `PtpIface`) et une interface vers le (les [a]) point(s) de terminaison de circuit attaché(s) (type `TtpIface`).

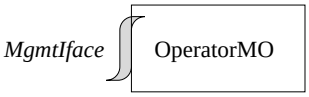


[a] Un point de terminaison physique peut héberger plusieurs points de terminaison de circuit

L'objet géré équipement (`EquipmentMO`) dispose d'une interface de gestion (type `MgmtIface`), et d'une interface vers les points de terminaison physique qu'il supporte (type `PtpIface`). Au travers de cette interface, il communiquera avec les points de terminaison afin de maintenir la cohérence de la relation de terminaison.



L'objet géré opérateur (`OperatorMO`) ne dispose que d'une interface de gestion (type `MgmtIface`). Au travers de cette interface, toutes les opérations de gestion sur les objets gérés pourraient être invoquées.



Les objets du modèle sont interconnectés au travers des interfaces définies ci-dessus. Cette interconnexion est illustrée dans la figure 2. On retrouve sur la figure le type des interfaces d'interconnexion.

Dans la suite de cette section, nous allons présenter la spécification LOBSTERS des interfaces des objets puis trois extraits de la spécification. Ces trois extraits de comportement correspondent à différentes étapes de l'exécution d'opérations sur la MIB pour la gestion de la configuration de l'interconnexion de commutateurs. Ces étapes sont:

1. le comportement standard dans lequel, l'ensemble des objets pour une configuration donnée sont instanciés,
2. l'attachement de circuit dans lequel on va instancier un circuit,
3. le détachement ou destruction d'un circuit et l'impact de cette opération sur le comportement des points de terminaison associés.

3.2 Les spécifications LOBSTERS des interfaces d'objets

Nous avons défini dans notre modèle, une seule spécification d'interface comportant l'ensemble des interactions possibles entre objets (voir figure 3). On y retrouve donc les opérations de gestion (`Lock, Unlock`) ainsi que les opérations annexes utiles pour la propagation des dépendances entre objets (`disable, enable` par exemple). Les interactions supplémentaires nous permettent de simuler la création et la destruction dynamique de circuits, celles-ci n'étant pas supportées dans le simulateur CRUSADE.

La figure 4 illustre la description LOBSTERS (GDMO étendu avec le concept d'interfaces et de comportements) des interfaces associées à un objet géré. Dans cet exemple, on voit que l'objet géré équipement est attaché à deux interfaces de type `PORTE`, l'une modélisant l'interface de gestion (interface avec l'opérateur), l'autre décrivant l'interface avec les points de terminaison physique éventuellement attachés.

L'ensemble des interfaces identifiées pour les objets (`TtpIface, MgmtIface, CsTpBelongIface, CsIface, CircuitIface, PtpIface, CsEndIface`) héritent de l'interface générique de type `PORTE`). Pour simplifer notre modèle, nous n'avons maintenu que le type `PORTE`.

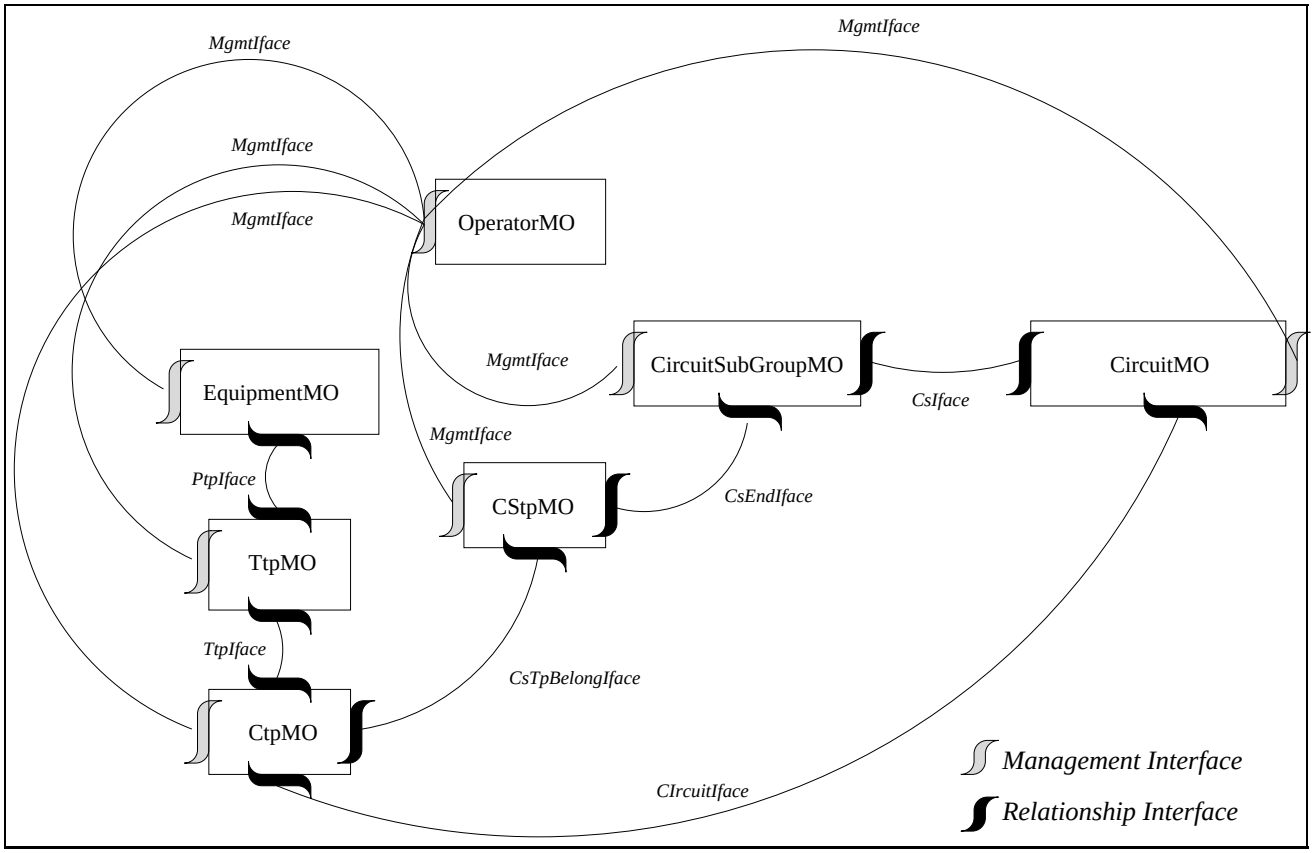


FIG. 2 – *Les objets LOBSTERS du modèle*

```
PORTE GATE
 DERIVED FROM SyncGate;
 DEFINITION " This interface is used to interconnect all objects";
 DECLARATIONS
    id     : INTEGER;
    id1    : INTEGER;
    id2    : INTEGER;

 EVENTS

    create_circuit(id,id1,id2);                    lock(id);
    delete_circuit(id);                            lockf(id);
    get_dependency(id);                            unlock(id);
    receive_dependency(id1,id2);                   disable(id);
    set_dependency(id,id1);                        enable(id);
    remove_dependency(id,id1);
;
```

FIG. 3 – *L'interface générique*

```
EquipmentMO MANAGED OBJECT CLASS
 DERIVED FROM Top;
 CHARACTERIZED BY EquipmentPackage;
 INTERFACES
          porte       : PORTE, // Management Interface
          porte_t_ttp : PORTE; // TTP Interface
 BEHAVIOUR EquipmentBehaviour;
 REGISTERED AS { lobsters-moc 1};
```

FIG. 4 – *Le squelette de l'objet équipement*

Dans l'environnement MODE, nous disposons d'un analyseur LOBSTERS, mais le générateur LOBSTERS ⇒ CRS n'est pas opérationnel à ce jour. Afin de permettre la simulation du modèle dans CRUSADE, nous avons directement écrit les spécifications en CRS. Ces spécifications sont données en annexe.

Nous allons maintenant présenter les comportements associés à ces objets. Pour cela, nous avons extrait des spécifications des exemples portant sur trois phases de la vie du modèle, à savoir le fonctionnement standard, la création d'un circuit ainsi que le détachement de celui-ci (sa destruction).

3.3 Le comportement standard

Dans une configuration standard, un circuit relie deux points de terminaison de circuit. Il est rattaché à un sous-groupe de circuits. Les deux points de terminaison de circuit sont eux liés chacun à un point de terminaison physique sur leur équipement respectif. De plus, ils sont attachés à des points de terminaison de sous-groupe de circuits. Les points de terminaison physique sont eux, liés à un équipement. Dans cette configuration, différentes opérations sont possibles. La figure 5 présente une telle configuration et les interfaces associées.

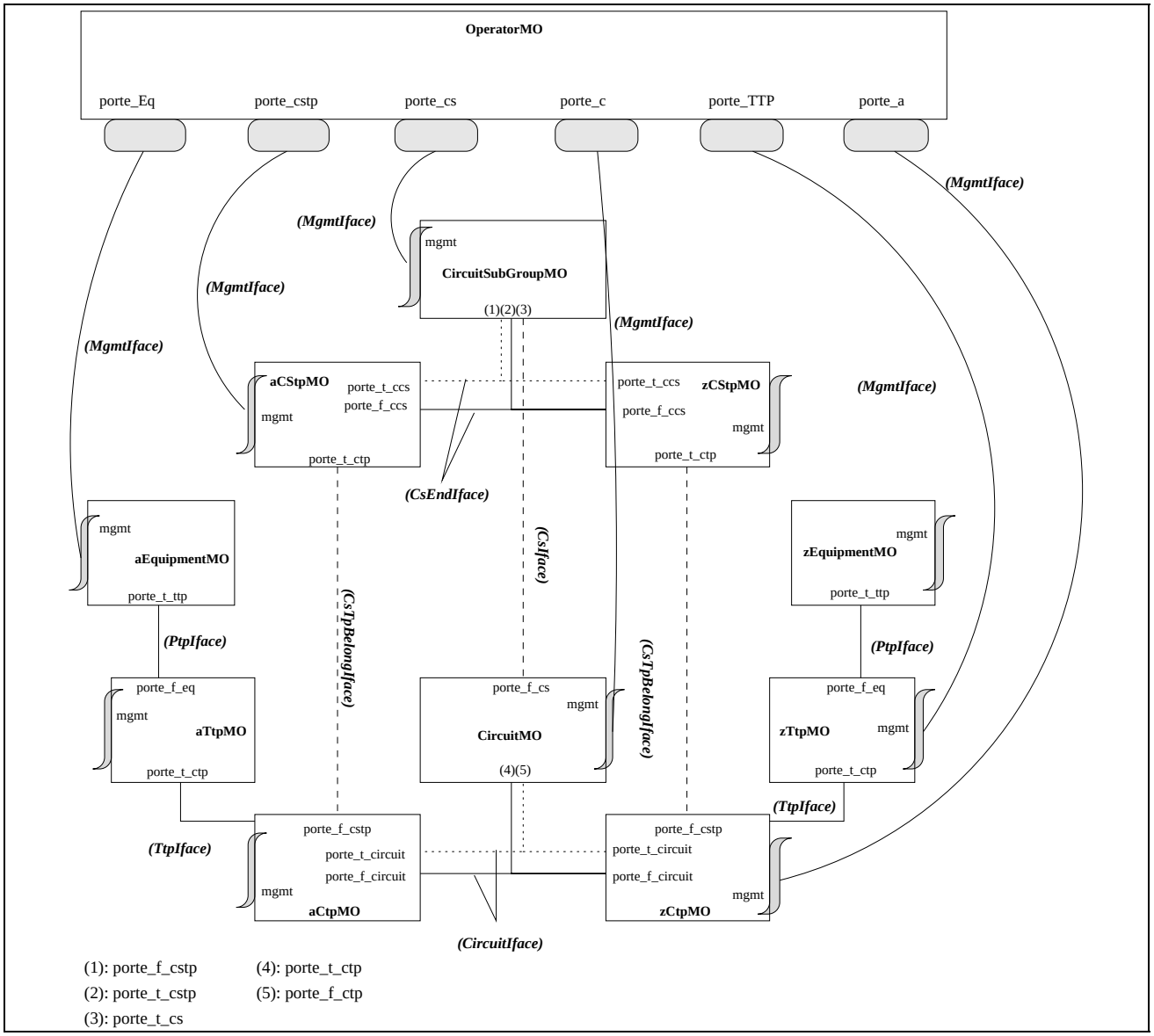


FIG. 5 – *La configuration CRS associée et les noms locaux des interfaces*

A toute interface à laquelle un objet est associé, celui-ci lui associe un nom local. Aussi la figure comporte les noms locaux aux objets des différentes instances d'interfaces (par exemple `mgmt` pour l'interface de gestion de l'équipement et `porte_t_ttp` pour l'interface avec les points de terminaison). Les opérations possibles sur une

telle configuration sont les opérations de gestion de base à savoir le blocage (`lock`), resp. déblocage administratif (`unlock`) d'un ou plusieurs objets du modèle. Nous allons dans cette section présenter les règles de comportement associées à ce type d'opération. Pour cela, nous allons présenter les deux vues comportementales sur la relation de terminaison `circuit/Ctp` à savoir celle du circuit et celle des points de terminaison du circuit.

3.3.1 La vue du circuit

La figure 6 comporte les deux règles de comportement associées au blocage (resp. déblocage) administratif sur un objet de type circuit (`CircuitMO`). Pour le blocage, la condition de déclenchement spécifie qu'il faut une invocation de l'opération `lock` à l'interface de gestion `porte` (ligne 2) et que l'identificateur d'objet corresponde à celui du circuit (ligne 3). La clause `EFFECTS` (4) spécifie les effets du blocage administratif d'un circuit. La première conséquence est le passage de l'état administratif à `locked` (5).

```
glock;                                          1
  COND: mgmt.lock(id)                           2
      AND identifier = id;                      3
  EFFECTS: administrativeState = locked         4
      AND vz_to_lock = 1                        5
      AND va_to_lock = 1                        6
      AND va_to_disable = 1                     7
      AND vz_to_disable = 1;                    8
```

```
unlock;                                         9
  COND: mgmt.unlock(id)                         10
      AND identifier = id;                      11
  EFFECTS: administrativeState = unlocked       12
      AND vz_to_unlock = 1                      13
      AND va_to_unlock = 1                      14
      AND vz_to_enable = 1                      15
      AND va_to_enable = 1;                     16
```

FIG. 6 – *Les opérations de blocage et déblocage administratif de circuit (comportement de* `CircuitMO`*)*

Les effets supplémentaires du blocage d'un circuit sont le besoin de:

- bloquer administrativement les deux points de terminaison (lignes 5,6).
- transformer l'état opérationnel des deux points de terminaison de circuit en le ramenant à `disable` et en ajoutant une dépendance à l'attribut `availabilityStatus`[3] (lignes 7,8).

Dans notre modèle, ces opérations se traduisent par l'affectation de variables car il n'est pas possible de déclencher plusieurs interactions sur une même interface de communication simultanément. Dans la règle `glock` l'on affecte ainsi quatre variables intermédiares qui vont servir à déclencher les échanges de messages au travers d'autres règles. Ces variables sont `vz_to_lock` (ligne 5) qui indique que l'on doit faire un `lock` sur le point `Z`; `va_to_lock` (ligne 6) qui indique que l'on doit faire un `lock` sur le point `A`; `va_to_disable` (ligne 7) qui précise le besoin de propager un `disable` vers le point `A` et finalement `vz_to_disable` (ligne 8) qui précise le besoin de propager un `disable` vers le point `Z`.

Nous avons donc pour chaque variable intermédiaire une règle correspondante dans le modèle qui spécifie l'échange de messages associé. Deux de ces règles de propagation sont données dans la figure 7.

```
az_lock_prop;                   1
COND: va_to_lock = 1;           2
EFFECTS: va_to_lock = 0         3
     AND porte_t_ctp.lock(ep)   4
     AND ep = 'aCtp;            5
```

```
az_disable_prop;                6
COND: va_to_disable = 1;        7
EFFECTS: va_to_disable = 0      8
     AND porte_t_ctp.disable(ep) 9
     AND ep = 'aCtp;            10
```

FIG. 7 – *Les règles de propagation de messages vers les points de terminaison*

La figure 7 contient la règle de propagation associée à l'envoi d'un message de blocage administratif (`lock`) (lignes 1 à 5) vers le point de terminaison A ainsi que celle de propagation d'un message de clôture de l'état opérationnel vers le même point de terminaison (lignes 6 à 10). Ces règles sont déclenchées par le positionnement de variables (`va_to_lock, va_to_disable`). Ces variables sont mises à jour dans la règle de blocage administratif du circuit. Le même type de règles existe dans la spécification pour le point de terminaison Z. Notons, de plus, que l'ordre d'exécution des propagations est ici aléatoire.

On peut également noter sur les opérations de blocage et de déblocage administratif de circuit que ces deux opérations sont parfaitement complémentaires.

3. Le passage à disable ou simplement une demande de passage à disable, si un objet est déjà dans cet état, entraîne automatiquement l'ajout d'une dépendances de type `Dependency` à l'attribut `availabilityStatus`.

En résumé, le blocage administratif d'un circuit se traduit par l'envoi de deux messages à chaque point de terminaison de celui-ci. Ces messages sont un ordre de blocage administratif ainsi qu'un ordre de blocage opérationnel. L'ordre de l'envoi ainsi que la priorité d'un point de terminaison par rapport à un autre sont aléatoires.

3.3.2 La vue des points de terminaison

Les points de terminaison sont susceptibles de recevoir des ordres de blocage des états administratifs et opérationnels provenant du circuit. Cela se traduit par deux règles présentées dans la figure 8.

```
lockProp;                                          1
COND: porte_f_circuit.lock(id)                     2
             AND id = identifier;                  3
EFFECTS: IF 'administrativeState = locked          4
         THEN TRUE                                 5
         ELSE administrativeState = locked         6
             AND propagate_disable = 1             7
         FI;                                       8
```

```
disable;                                                          9
COND: porte_f_circuit.disable(id)                                 10
    AND id = identifier;                                          11
EFFECTS: operationalState = disabled                              12
 AND availabilityStatus = 'availabilityStatus + 1;                13
                                                                  14
issue_disable;                                                    15
COND: propagate_disable = 1;                                      16
EFFECTS: propagate_disable = 0                                    17
      AND porte_t_circuit.disable(id)                             18
      AND id = identifier;                                        19
```

FIG. 8 – *Les règles de propagation de message dans les points de terminaison*

Les règles de comportement sont au nombre de trois. La première décrit la réaction d'un point de terminaison de circuit lors de la réception d'un ordre de blocage administratif provenant du circuit associé (règle `lockProp`). La condition associée à cette règle est l'occurrence d'un événement de type `lock` à l'interface avec le circuit (ligne 2) et l'égalité de l'identificateur avec celui-du point de terminaison (ligne 3). Les effets associés à cette règle déclenchent la propagation d'une dépendance dans le sens inverse (vers le circuit) si le point de terminaison n'était pas dans l'état administratif bloqué avant la réception de l'ordre (lignes 6,7). Si il était dans cet état (bloqué administrativement, l'ordre n'aurait eu aucun impact sur l'objet (lignes 4,5). La propagation d'une dépendance se fait au travers de la règle `issue_disable` (lignes 15 à 19).

La réception d'un ordre de blocage opérationnel (lignes 9 à 13) implique l'incrémentation du degré de dépendance (ligne 13) et le maintient ou le passage de l'état opérationnel à `disabled` (ligne 12).

Sur la base de ces comportements on identifie la possibilité de repropager des dépendances `disable` vers le circuit lorsque le point de terminaison voit sont état administratif passer à `locked`. De plus, la spécification comporte les règles inverses qui décrivent les conséquences sur le circuit et les points de terminaison du déblocage administratif du circuit et des points de terminaison.

3.4 L'attachement de circuit

La provision ou l'attachement d'un circuit correspond à une étape de la configuration d'un système d'interconnexion. La mise en place d'un circuit requiert l'existence de deux points de terminaison (aucune contrainte n'est donnée dans la norme sur leur état avant la provision). Tout comme pour le fonctionnement normal, nous allons ici illustrer les comportements associés au circuit et aux deux points de terminaison de circuit associés.

Dans notre spécification, nous somme partis du principe que l'attachement à un sous-groupe de circuit était automatique et que ce dernier était dans un état opérationnel actif et administratif non bloqué. Toute autre configuration est spécifiable sans problèmes en LOBSTERS mais entraîne de nouvelles règles de comportement que nous n'avions pas prises en compte dans notre précédent modèle. La figure 9 illustre le schéma de mise en place de circuit dans notre configuration.

Le comportement associé à cette activité va principalement devoir traiter la mise à jour des attributs d'état opérationnel et administratif des objets nouvellement impliqués dans une relation de type terminaison. En effet, l'attachement d'un circuit à deux points de terminaison entraîne des propagations de comportement qui ne se faisaient pas entre les deux points de terminaison avant cette connexion. Regardons ce comportement suivant les deux points de vue du circuit et du point de terminaison.

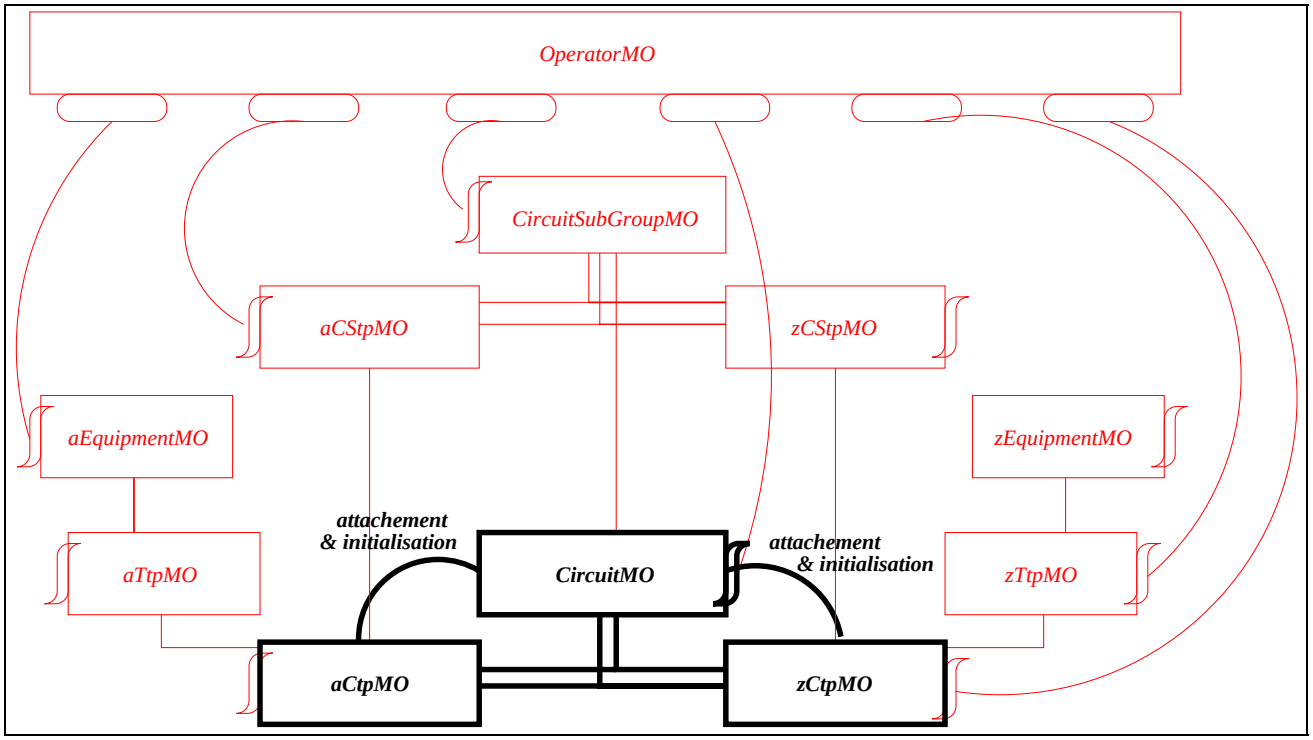


FIG. 9 – *La configuration d'attachement de circuit*

3.4.1 La vue du circuit

Comme la norme ne donne aucune information sur cette mise en place si ce n'est que de dire que l'objet est créé et administrativement bloqué, nous avons décidé de définir formellement la manière dont les attributs d'état du circuit et ceux des points de terminaison associés sont affectés lors de l'initialisation du circuit.

D'après notre définition du comportement de circuit lors de son initialisation, celui-ci va récupérer les degrés de dépendances dans les deux points de terminaison et propager ceux venant de A vers Z et inversement. Il va donc, dans un premier temps, harmoniser les dépendances sur sa relation de terminaison. Dans une deuxième étape de l'initialisation, le circuit va bloquer administrativement les deux points de terminaison car la norme précise qu'à l'initialisation, le circuit est administrativement bloqué. Si le blocage administratif des points de teminaison est justifié par le fait que le circuit est lui-même initialement administrativement bloqué, la propagation des dépendances à l'initialisation mérite quelques explications.

Lorsqu'un circuit n'est pas instancié et que deux points de terminaison ne sont donc pas liés, ils peuvent rester actifs au sens où ils peuvent exister, être liés à des points de terminaison physique et donc à des équipements. Cette existence peut faire apparaître dans l'un ou l'autre de ces points des dépendances opérationnelles (par exemple si le point de terminaison physique ou l'équipement auquel est attaché un point de terminaison de circuit est bloqué). Or l'absence de circuit ne permet pas de propager ces dépendances car les points de terminaison de circuit ne sont pas liés. Si, à l'initialisation du circuit, les dépendances ne sont pas propagées, le système peut dériver dans un état incohérent (par exemple un ciruit opérationnel et débloqué reliant deux équipements dont tous les points de terminaison sont inopérants).

Propager les dépendances à l'initialisation du circuit se traduit donc de la manière suivante:

1. lecture de l'état opérationnel et de l'état administratif des points de terminaison A et Z;
2. propagation du nombre de dépendances de l'attribut `availabilityStatus - 1`[4] du point de terminaison A vers Z et incrémenter l'attribut `availabilityStatus` du circuit du même nombre;
3. appliquer la même opération de Z vers A;

4. La motivation pour le retrait d'un degré de dépendance ainsi que la garantie que tout point de terminaison possède au moins une dépendance est donnée dans la section suivante sur le retrait d'un circuit. On y verra que le retrait d'un circuit tout comme l'état initial d'un point de terminaison engendre un état opérationnel de celui-ci à `disabled` et l'ajout d'une dépendance.

4. si l'état administratif de A est à `locked` alors propager une dépendance vers Z et incrémenter l'attribut `availabilityStatus` du circuit de 1;

5. appliquer la même opération de Z vers A;

Cette partie de l'algorithme d'initialisation se traduit dans le comportement formel en un ensemble de règles définies dans la figure 10.

```
create_circuit;
COND: mgmt.create_circuit(id,ep,ep1);
EFFECTS: aCtp = ep
     AND zCtp = ep1
     AND geta = 1
     AND getz =1;



getadep;
COND: geta =1;
EFFECTS: id = identifier
     AND ep = 'aCtp
     AND geta = 0
     AND initStep = 'initStep + 1
     AND porte_t_ctp.get_dependency(ep);

setaDep;
COND: NOT(toa = 1)
  AND geta = 0;
EFFECTS: porte_t_ctp.set_dependency(id,ep1)
     AND initStep = 'initStep + 1
     AND ep1 = 'toa - 1
     AND toa = 1
     AND id = 'aCtp;

receive_adep;
COND: porte_f_ctp.receive_dependency(ept,ep1)
  AND ept = aCtp;
EFFECTS:
 availabilityStatus = 'availabilityStatus + ep1 - 1
     AND fromA = ep1 - 1
     AND IF NOT (ep1  = 1)
         THEN toz = ep1
           AND initStep = 'initStep + 1
         ELSE initStep = 'initStep + 2
         FI;
```

```
1  terminate_init;
2  COND: initStep = 6;
3  EFFECTS: administrativeState = locked
4        AND vz_to_lock = 1
5        AND vz_to_disable = 1
6        AND va_to_disable = 1
7        AND va_to_lock = 1
8        AND releaseStep = 0
9        AND attached =1;
10
11 getzdep;
12 COND: getz =1;
13 EFFECTS: id = identifier
14       AND ep = 'zCtp
15       AND getz = 0
16       AND initStep = 'initStep + 1
17       AND porte_t_ctp.get_dependency(ep);
18
19 setzDep;
20 COND: NOT(toz = 1)
21  AND getz = 0;
22 EFFECTS: porte_t_ctp.set_dependency(id,ep1)
23       AND initStep = 'initStep + 1
24       AND ep1 = 'toz - 1
25       AND toz = 1
26       AND id = 'zCtp;
27
28 receive_zdep;
29 COND: porte_f_ctp.receive_dependency(ept,ep1)
30   AND ept = zCtp;
31 EFFECTS:
32  availabilityStatus = 'availabilityStatus + ep1 - 1
33   AND fromZ = ep1 - 1
34   AND IF NOT (ep1  = 1)
35       THEN toa = ep1
36        AND initStep = 'initStep + 1
37       ELSE initStep = 'initStep + 2
38       FI;
```

FIG. 10 – *Le comportement d'initialisation de circuit.*

L'ensemble des étapes d'initialisation du circuit est défini au travers de 8 règles. La première (`create_circuit` ligne 1) déclenche le processus d'initialisation. Cette règle est déclenchée lorsque le circuit reçoit de l'opérateur un ordre de création qui comporte l'identificateur du circuit ainsi que les identificateurs des deux points de terminaison qui lui sont affectés. Cette opération a pour effets l'affectation des pointeurs de point de terminaison (lignes 3 et 4) ainsi que le démarrage du processus d'acquisition des dépendances (lignes 5,6).

L'ordre d'acquisition est aléatoire et cette acquisition est réalisée au travers des règles `get_XXXdep` et `receive_XXXdep` où `XXX` représente le rôle d'un point de terminaison (`A` ou `Z`). Lors de la réception des dépendances (`receive_XXXdep`), la valeur de l'attribut `availabilityStatus` du circuit est incrémentée du degré de dépendance reçu -1.

La seconde étape consiste à propager les dépendances reçues d'un rôle de point de terminaison vers l'autre. Ceci est réalisé au travers des règles `setXXXdep` (lignes 19-25 et 58-64). Ici, on propage le nombre de dépendances du point de terminaison `A` vers `Z` et inversement. Cette opération se produit nécessairement après l'acquisition des dépendances.

La dernière étape du processus d'initialisation de la relation de terminaison de circuit (règle **terminate_init** lignes 39-47) positionne l'état administratif du circuit à **locked**, déclenche la procédure de blocage administratif et opérationnel des points de terminaison (lignes 42 à 47), indique que le circuit est attaché et qu'il peut être détaché après la propagation des blocages administratifs.

3.4.2 La vue du point de terminaison

Le point de terminaison doit, pour permettre l'initialisation, offrir une méthode de lecture de ses états ainsi qu'une méthode d'affectation de ceux-ci. Nous avons également étendu l'objet point de terminaison avec un attribut spécifiant si un circuit est ou non attaché à un moment donnée. Cela nous permet dans les règles de description du comportement standard, de lier les propagations à la présence d'un circuit. Cependant, le point de terminaison ne connait pas quel circuit est attaché (son identificateur par exemple) mais seulement si un circuit est lié. La reconnaissance entre le circuit et le point de terminaison ne se fait que dans le circuit. C'est pourquoi le point de terminaison inclut son identificateur (**identifier**) dans tout message qu'il envoie vers un circuit. Ce dernier décide si cet identificateur correspond à l'un des points de terminaison auxquels il est rattaché de traiter le message ou non.

Pour l'ordre de blocage administratif, le comportement est identique à celui d'un ordre venant de l'interface de gestion. Nous avons, pour la lecture et l'affectation des degrés de dépendance à l'initialisation d'un ciruit, ajouté deux règles au comportement d'un point de terminaison. Ces règles sont données dans la figure 11.

```
getdep;                                                    1
COND: porte_f_circuit.get_dependency(id)                   2
      AND identifier = id;                                 3
EFFECTS:id1 = identifier                                   4
 AND IF 'administrativeState = locked                      5
     THEN cid = 'availabilityStatus + 1                    6
     ELSE cid = 'availabilityStatus                        7
     FI                                                    8
 AND availabilityStatus = 'availabilityStatus -1           9
 AND IF availabilityStatus = 0                             10
     THEN operationalState = enabled                       11
     ELSE operationalState = disabled                      12
     FI                                                    13
 AND attachedCircuit = 1                                   14
 AND porte_t_circuit.receive_dependency(id1,cid);          15
```

```
setdep;                                                    16
COND: porte_f_circuit.set_dependency(id,ep)                17
      AND identifier = id;                                 18
EFFECTS:                                                   19
  availabilityStatus = 'availabilityStatus + ep;           20
```

FIG. 11 – *Les règles d'attachement de circuit dans un point de terminaison*

La première règle (**getdep**) traite la demande du nombre de dépendances venant d'un circuit. Les effets associés à cette demande sont l'attachement du circuit (ligne 14), le retrait d'une dépendance, le passage éventuel de l'état opérationnel à **enabled** (lignes 9-13) et une réponse comportant le nombre de dépendances $+$ ou -1 si l'état administratif est déjà bloqué ou non (lignes 5 à 8). La seconde règle (**setdep**) permet de positionner les dépendances propagées.

3.5 Le détachement de circuit

Le principal problème qui se pose lors de la destruction d'un circuit concerne la remise en état des états des deux points de terminaison qui le lient. Sur ce point, la norme reste muette. Elle précise cependant que cette opération a des implications sur les états des points de terminaison mais n'en précise pas les modalités.

Il nous a donc fallu sur la base des relations que nous avons identifiées, définir ces comportements et les spécifier de manière formelle en LOBSTERS. Les définitions comportementales que nous avons définies sont les suivantes:

- la destruction d'un circuit n'entraîne pas la destruction des points de terminaison associés;
- la destruction d'un circuit entraîne la mise à l'état **disabled** des points de terminaison et ajoute une dépendance à chacun de ces points[5];

5. Ceci permet de ne pas pouvoir utiliser un point de terminaison tant qu'aucun circuit n'y est attaché.

– la destruction d'un circuit annule toutes les propagations de dépendances effectuées durant son existence sur les points de terminaison;

– la destruction d'un circuit bloque administrativement les deux points de terminaison.

Cette définition de comportement se traduit dans la spécification en un ensemble de règles au niveau du circuit et des points de terminaison. Ces règles sont présentées ci-dessous.

3.5.1 Le point du vue du circuit

Le circuit reçoit une opération de détachement provenant de l'opérateur. Cette opération déclenche trois règles qui se chargent de la mise à jour des attributs des points de terminaison. Ces règles sont spécifiées dans la figure 12.

```
delete_circuit;                                          1
COND: mgmt.delete_circuit(id)                            2
  AND id = identifier;                                   3
EFFECTS: seta = 1                                        4
   AND releaseStep = 1                                   5
   AND setz = 1                                          6
   AND vz_to_lockf = 1                                   7
   AND va_to_lockf = 1;                                  8
                                                         9
setzDep1;                                                10
COND: setz = 1;                                          11
EFFECTS: porte_t_ctp.remove_dependency(id,ep1)           12
     AND releaseStep = 'releaseStep + 1                  13
     AND ep1 = 'fromA                                    14
     AND setz = 0                                        15
     AND id = 'zCtp;                                     16
```

```
terminate_release_p;                                     17
COND: releaseStep = 3;                                   18
EFFECTS: attached = 0                                    19
     AND initStep = 0;                                   20
                                                         21
                                                         22
                                                         23
                                                         24
                                                         25
setaDep;                                                 26
COND:seta = 1;                                           27
EFFECTS: porte_t_ctp.remove_dependency(id,ep1)           28
     AND releaseStep = 'releaseStep + 1                  29
     AND ep1 = 'fromZ                                    30
     AND seta = 0                                        31
     AND id = 'aCtp;                                     32
```

FIG. 12 – *Le comportement de détachement de circuit.*

La première règle (`delete_circuit`) est déclenchée sur l'occurrence d'un événement de détachement de circuit (lignes 2-3). Cette opération a pour effet de permettre le déclenchement des règles de positionnement des dépendances ainsi que des règles de blocage administratif final des points de terminaison. Les règles de positionnement de dépendance (`setXXXDep` lignes 10-16 et 27-32) envoient aux points de terminaison le nombre de dépendances à enlever. Celui-ci correspond au nombre de dépendances propagées durant la vie du circuit. Le détachement se termine par l'annulation de l'indicateur d'attachment.

3.5.2 La vue du point de terminaison

Lors de la réception d'un ordre de repositionnement du degré de dépendance, le point de terminaison décrémente son attribut de dépendance (`availabilityStatus`) du nombre de dépendances indiquées par le circuit.

```
removedep;
COND: porte_f_circuit.remove_dependency(id,ep)
      AND identifier = id;
EFFECTS: availabilityStatus = 'availabilityStatus - ep;

lockf;
COND: porte_f_circuit.lockf(id)
    AND id = identifier;
EFFECTS: administrativeState = locked
    AND availabilityStatus = 'availabilityStatus +1
    AND operationalState = disabled
    AND attachedCircuit = 0;
```

FIG. 13 – *Le comportement de détachement de circuit vu par le point de terminaison.*

Puis, lors de la réception du message de blocage administratif final, son état administratif passe à `locked`, le nombre de dépendances est incrémenté de 1 et son état opérationnel passe à `disabled` La figure 13 montre les règles associées à ce comportement.

4 Résultats obtenus

4.1 Résultats généraux

4.1.1 De l'utilité de LOBSTERS

Cette étude a confirmé l'adéquation de LOBSTERS aux besoins de spécification des comportements des objets gérés en administration de réseaux. Elle a prouvé qu'il est possible d'exprimer sans trop d'efforts les dépendances comportementales en LOBSTERS. Construire une spécification LOBSTERS oblige le spécifieur à bien identifier les relations entre les objets ce qui est déjà un apport important et largement positif. Ces relations, le spécifieur va ensuite devoir les traduire sous forme de d'interfaces de communication supportant un ensemble de messages et spécifier un certain nombre de règles de comportement au sein des objets qui prennent un ou plusieurs rôles dans une relation. Le résultat de cette étape de modélisation est une spécification complète, claire dont l'exactitude et la cohérence reste à valider. Pour cette étape de validation nous avons utilisé le simulateur CRUSADE qui nous a permis au travers de la simulation interactive de valider notre modèle, i.e. vérifier le fonctionnement espéré du système.

Cette étude nous a donc permis de valider les concepts définis dans LOBSTERS sur un nouveau cas d'étude réel. Elle a de plus montré l'intérêt du support ASN.1 dans la description formelle qui a permis de ne pas s'éloigner de la spécification initiale fournie en GDMO.

4.1.2 Limites des outils

Ce travail nous a également permis d'identifier un certain nombre de limites de l'implantation du simulateur, limites que nous avons contournées en modifiant certains points de la spécification. Cela a impliqué une croissance non négligeable en nombre de lignes de la spécification. Ceci est principalement dû au non-support dans les outils, notamment CRUSADE, de la possibilité d'émettre simultanément (au sein d'un même `EFFECTS` plusieurs messages au travers d'un ou plusieurs canaux de communication. Cela nous a obligé à découper les comportements impliquant plusieurs échanges de messages en autant de règles que de messages échangés (cas du `lock` d'un équipement ou d'un circuit par exemple). Ceci a nécessité l'introduction de nombreuses variables locales. En résumé, sur le plan des outils, des travaux restent à faire autour de LOBSTERS.

4.1.3 Quelques pistes

La standardisation d'un modèle géneral de description des relations entre objets gérés fournit une base pour la description des dépendances et des comportements inter-objets. Comme LOBSTERS est initialement basé sur les objets eux-mêmes, l'utilisation de cette nouvelle notation semi-formelle ouvre de nouvelles pistes pour la spécification des comportements. Entre autres, nous aimerions pouvoir "sortir" la partie comportement liée à une relation de celle spécifique à un objet. De même que nous envisageons la génération automatique d'une partie de l'infrastructure (interfaces de communication , messages, règles de comportement) à partir de la spécification plus abstraite des dépendances. Cela se rapproche notamment des travaux liés au modèle de référence ODP [ODP1 92, ODP2 92, ODP3 92] où l'on désire passer d'un modèle dit de l'information (classes + dépendances) à celui de traitement (objets + interactions).

4.2 Liens avec les autres études

L'approche présentée ici diffère de celle entreprise avec les systèmes de transitions étiquetées et le logiciel MEC. Dans cette première étude, nous n'avons pu vérifier qu'un sous-ensemble du modèle à savoir le comportement du système en conditions normales (tous les objets sont instanciés et connectés entre eux). Sur cette configuration initiale, nous avons pu vérifier la correction de notre extension du modèle après avoir identifié grâce à l'outil des erreurs dans la spécification normative.

Pour la spécification LOBSTERS, nous sommes partis des nouvelles règles de comportement vérifiées avec MEC pour une configuration standard. Cette nouvelle approche et les fonctionnalités du LOBSTERS nous ont permis, d'une part de simuler interactivement le modèle, d'autre part d'aborder d'autres aspects du modèle comme l'attachement et le détachement de circuit. Le grand nombre d'interactions possible ne nous permet pas

d'affirmer que tous les scénarios possibles ont été simulés mais la simulation a été très utile pour comprendre le fonctionnement global du modèle et "débugger" la spécification.

La solution idéale pour une validation complète du système serait de permettre la génération d'un automate compatible avec MEC à partir de la spécification LOBSTERS et de valider des scénarios préalablement simulés. Une étude de faisabilité est actuellement en cours sur ce point et il semble que cela soit réalisable sur un sous-ensemble du langage LOBTSERS.

Le second point intéressant de cette étude concerne l'approche différente des relations (canaux de communication + échanges + comportement) que nous avons eue. Cela nous a permis de spécifier le système en construisant non plus d'une pièce l'ensemble du système et des propagations associées (comme cela vait été réalisé en MEC dans le vecteur de synchronisation) mais toujours avec la vue d'un objet et des seules relations dans lesquelles il est impliqué. Ceci nous a permis de se rapprocher du système réel tout en restant à un niveau d'abstraction suffisant pour le décrire de manière assez simple.

C'est sur cette voie que nous voulons continuer nos investigations.

5 Conclusion et perspectives

Dans ce rapport, nous avons présenté l'utilisation de `LOBSTERS` pour la spécification du modèle de gestion de la configuration de l'interconnexion des commutateurs. Nous avons montré que l'approche LOBSTERS permettait de spécifier de manière formelle les comportements d'objets au sein d'une base d'information de gestion.

Le principal enseignement que nous pouvons tirer des deux études menées sur le modèle de gestion de la configuration de l'interconnexion de commutateurs est que quelque soit la méthode formelle utilisée, sa mise en œuvre permet au moins de diminuer le nombre d'erreurs ou d'oublis dans un modèle et permettra toujours de clarifier la spécification même informelle de celui-ci. En effet, dans les deux spécifications que nous avons établies, nous avons mis l'accent sur différentes propriétés du systèmes, ce qui nous a obligé lors de la conception de la spécification à nous poser des questions sur le modèle, questions auxquelles nous n'avons pour la plupart du temps trouvé aucune réponse dans la norme, trop peu précise. Vu la taille des spécifications résultantes, toute vérification sans support logiciel aurait été difficile et hasardeuse. Sur ce point, le logiciel `MEC` a été très utile tout comme le simulateur CRUSADE (même si tous les scénarios n'ont pas été simulés).

L'étape suivante de nos travaux va se concentrer sur la manière d'exploiter les spécifications des relations à l'aide du GRM pour la génération automatique des points d'interaction entre objets. De plus, une étude du modèle présenté ici avec le langage LDS [CCITT-Z-100-92 92] est également planifiée. Celle-ci devrait cependant être assez proche de la conception LOBSTERS.

Références

[Arnold 89] A. Arnold. *"MEC: a system for constructing and analysing transition systems"*. 1989. Proc Int. Workshop on Automatic Verification Methods for Finite State Systems.

[CCITT-Z-100-92 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Specification and Description Language (SDL) 1992"*, Norme Internationale, CCITT-Z-100-92, Janvier 1992.

[CCITT.X.722 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Technologies de l'Information - Interconnexion de Systèmes Ouverts - Structure des Informations de Gestion: Directives por la Définition des Objets Gérés "*, Norme Internationale, CCITT.X.722, Janvier 1992, [ISO-10165.4 92].

[Eschebach 91] Wilko Eschebach. *"Interpretative Ausfuehrung kommunizierender Regelsysteme"*. 1991. Master Thesis, University of Kaiserslautern, Germany, 1991.

[Festor 93] O. Festor et G. Zoerntlein. *"Formal Description of Managed Object Behavior - A Rule Based Approach"*. pages 45–58, 1993. in [ISINM'93].

[Festor 94a] O. Festor. *"Formalisation du comportement des objets gérés dans le cadre du modèle OSI"*. Ph.D. Thesis, University H.Poincaré-Nancy I, France, 1994.

[Festor 94b] O. Festor. *"OSI Managed Objects Development with LOBSTERS"*. 1994. Fifth International Workshop on Distributed Systems: Operations and Management, 12-16 Septembre 1994, Toulouse, France.

[Festor 94c] O. Festor et Schaff A. *"Comportement des objets gérés: une approche formelle"*. 1994. Colloque Francophone sur l'Ingenierie des Protocoles CFIP'95, Rennes France.

[Festor 95] O. Festor. *"MODE: a Development Environment for Managed Objects"*. pages 616–628, 1995. in [ISINM'95].

[Frot 93] J.P. Frot, H. Lecorguillé, J. Lefranc et D. Orain. *"CRUSADE: un environnement de développement de protocoles"*. 1993. Industrial Project Report, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

[ISINM'93] ISINM'93. *Integrated Network Management, III (C-12).* Elsiever Science Publishers B.V. (North-Holland). Proc. IFIP IEEE 3rd Int. Symp. on Integrated Network Management, San Francisco, CA, 18-23 April, 1993.

[ISINM'95] ISINM'95. *Integrated Network Management, IV.* Chapman & Hall. Proc. IFIP IEEE 4th Int. Symp. on Integrated Network Management, Santa Barbara, CA, 1-5 May, 1995.

[ISO-10165.4 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects"*, International Standard, ISO-10165.4, January 1992.

[ISO-10165.5 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 5: Generic Management Information"*, International Standard, ISO-10165.5, January 1992.

[Mackert 87] L.F. Mackert et I.B. Neumeier-Mackert. *"Communicating Rule Systems"*. pages 77–88, 1987. Proc. 7th. Int. Symp. on Protocol Specification, Testing and Verification, H. Rudin, C.H. West (editors), North-Holland 1987.

[Nataf 96] E. Nataf, O. Festor et A. Schaff. *"Validation du modèle de gestion d'interconnexion de commutateurs à l'aide de systèmes de transitions étiquetées"*. January 1996. INRIA Research Report Nr.???, (submitted).

[NMF 94] Network Management Forum, *"Switch Interconnection Management: Configuration Management Ensemble"*, NMF, November 1994.

[ODP1 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Draft Recommendantion X.901: Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use"*, Committed Draft, ODP1, November 1992.

[ODP2 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Draft Recommendantion X.901: Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model"*, Committed Draft, ODP2, November 1992.

[ODP3 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Draft Recommendantion X.901: Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model"*, Committed Draft, ODP3, November 1992.

[Orain 93] D. Orain. *"A New ASN.1 Compiler for the CRUSADE Environment"*. 1993. Master Thesis, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

[Schneider 92] J.M. Schneider. *"Protocol Engineering: A Rule-based Approach"*. Vieweg, 1992.

6 Annexes

6.1 Un écran CRUSADE

La figure 14 comporte une copie d'écran du simulateur CRUSADE lors d'une session de simulation interactive du modèle de gestion de la configuration d'interconnexion de commutateurs. La configuration testée ici comporte un opérateur, deux équipements (`eqAMO, eqZMO`), deux points de terminaison physique liés chacun à un équipement (`aTTPMO, zTTPMO`), deux points de terminaison de circuits (`aCtpMo, zCtpMO`), deux points de terminaison de sous-groupe de circuits (`aCsTpMO, zCsTpMO`), un sous-groupe de circuits (`subgroupMO`) et une instance de circuit (`circuitMO`). Cette configuration correspond au scénario défini dans l'annexe 6.2.9.

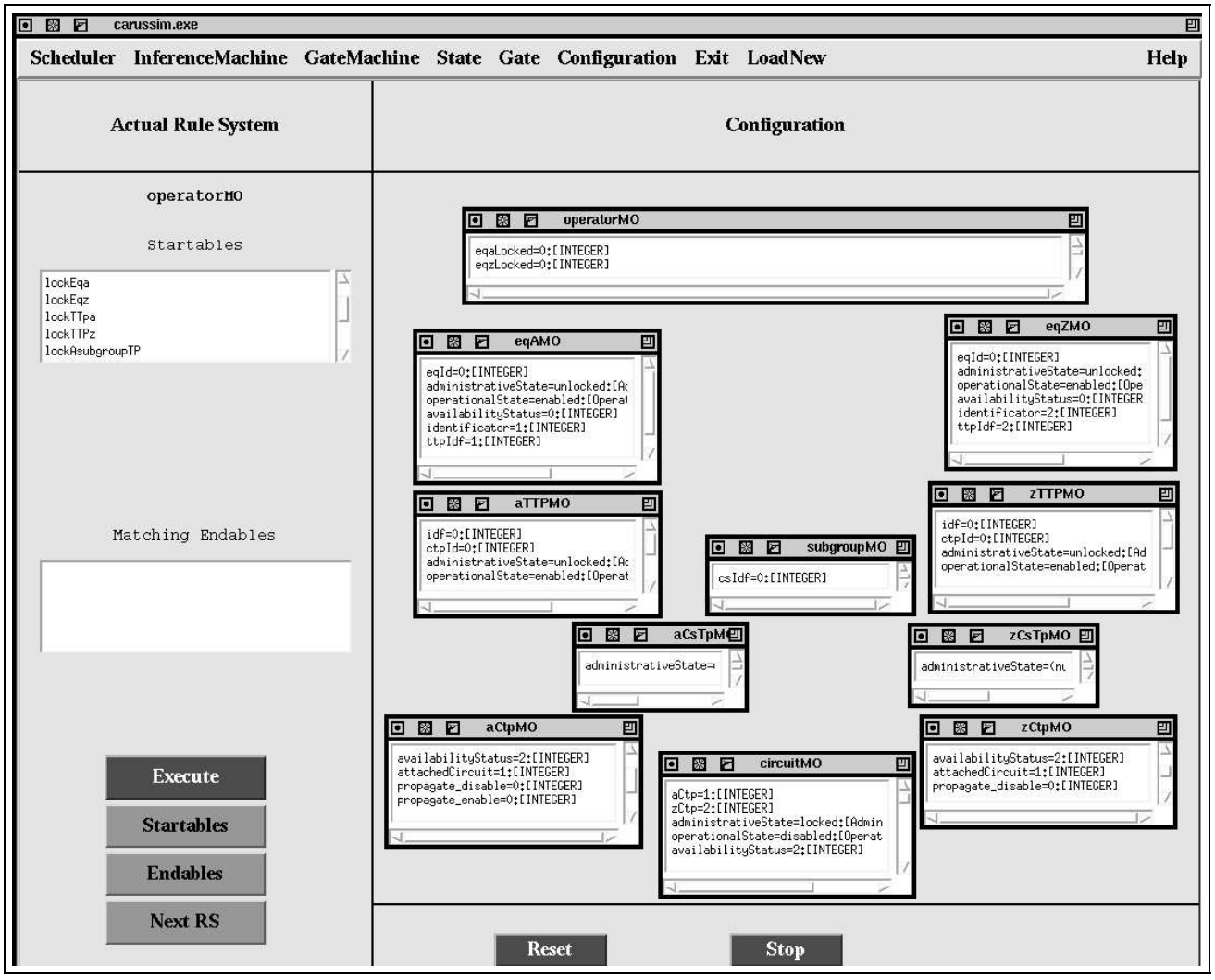


FIG. 14 – *Une configuration sous CRUSADE*

La copie d'écran a été prise après instanciation du circuit. On peut voir que celui-ci est à l'état administratif `locked`, opérationnel `disabled` et que deux dépendances sont présentes dans son attribut `availabiityStatus`. Les deux points de terminaison sont également administrativement bloqués et à l'état opérationnel `disabled`. Ils affichent dans leur attribut `availabilityStatus` deux dépendances propagées, l'une par le point opposé, l'autre par le blocage administratif du circuit. Les règles déclenchables visibles à gauche de l'écran, sont une partie de celles déclenchables par l'opérateur à ce moment de la simulation.

6.2 La spécification CRS complète

Comme présenté dans les section précédentes de ce rapport, toute spécification LOBSTERS se traduit en une spécification CRS qui elle est directement utilisable par les outils de simulation. Le générateur LOBSTERS vers CRS n'étant pas opérationnel dans MODE à ce jour, nous avons réalisé une partie de la description directement en CRS. Cette spécification est donnée ci-dessous.

6.2.1 Les interfaces de communication

La spécification de l'interface de communication standard est donnée dans la section 3.2. Toutes les interfaces du modèle en dérivent. La traduction en CRS de cette interface est donnée ci-dessous.

```
GATE SYNCHRONOUS PORTE;
DECLARATIONS

    id     : INTEGER;
    id1    : INTEGER;
    id2    : INTEGER;

EVENTS

    create_circuit(id,id1,id2);
    delete_circuit(id);
    get_dependency(id);
    receive_dependency(id1,id2);
    set_dependency(id,id1);
    remove_dependency(id,id1);
    lock(id);
    lockf(id);
    unlock(id);
    disable(id);
    enable(id);

END_GATE PORTE;
```

6.2.2 L'opérateur

```
RULE_SYSTEM OPERATOR ( porte_Eq    : PORTE;
                       porte_TTP   : PORTE;
                       porte_a     : PORTE;
                       porte_c     : PORTE;
                       porte_cstp  : PORTE;
                       porte_cs    : PORTE);

STATE

eqaLocked             : INTEGER INITIALLY 0;
ttpaLocked            : INTEGER INITIALLY 0;
actpLocked            : INTEGER INITIALLY 0;
circuitLocked         : INTEGER INITIALLY 0;
cSTPALocked           : INTEGER INITIALLY 0;
cSLocked              : INTEGER INITIALLY 0;

eqzLocked             : INTEGER INITIALLY 0;
ttpzLocked            : INTEGER INITIALLY 0;
zctpLocked            : INTEGER INITIALLY 0;
circuitInstalled      : INTEGER INITIALLY 0;
cSTPZLocked           : INTEGER INITIALLY 0;

DECLARATIONS
  id : INTEGER;
  ep : INTEGER;
  ep1 : INTEGER;

RULES

lockEqa;
 COND: eqaLocked = 0;
 EFFECTS: eqaLocked = 1
      AND id = 1
      AND porte_Eq.lock(id);

unlockEqa;
 COND: eqaLocked = 1;
 EFFECTS: eqaLocked = 0
      AND id = 1
      AND porte_Eq.unlock(id);

lockTTpa;
 COND: ttpaLocked = 0;
 EFFECTS: ttpaLocked = 1
      AND id = 1
      AND porte_TTP.lock(id);

unlockTTPa;
 COND: ttpaLocked = 1;
 EFFECTS: ttpaLocked = 0
      AND id = 1
      AND porte_TTP.unlock(id);

lockACTP;
COND: actpLocked = 0;
EFFECTS: actpLocked = 1
         AND id = 1
         AND porte_a.lock(id);

lockEqz;
 COND: eqzLocked = 0;
 EFFECTS: eqzLocked = 1
      AND id = 2
      AND porte_Eq.lock(id);

unlockEqz;
 COND: eqzLocked = 1;
 EFFECTS: eqzLocked = 0
      AND id = 2
      AND porte_Eq.unlock(id);

lockTTPz;
 COND: ttpzLocked = 0;
 EFFECTS: ttpzLocked = 1
      AND id = 2
      AND porte_TTP.lock(id);

unlockTTPz;
 COND: ttpzLocked = 1;
 EFFECTS: ttpzLocked = 0
      AND id = 2
      AND porte_TTP.unlock(id);

lockZCTP;
COND: zctpLocked = 0;
EFFECTS: porte_a.lock(id)
      AND id = 2
         AND zctpLocked = 1;
```

```
   unlockACTP;
   COND: actpLocked = 1;
   EFFECTS: porte_a.unlock(id)
         AND id = 1
            AND actpLocked = 0;

   lockAsubgroupTP;
   COND: cSTPALocked = 0;
   EFFECTS: cSTPALocked = 1
         AND id = 1
            AND porte_cstp.lock(id);

   unlockAsubgroupTP;
   COND: cSTPALocked = 1;
   EFFECTS: porte_cstp.unlock(id)
            AND id = 1
            AND cSTPALocked = 0;

   lockCircuit;
   COND: circuitLocked = 0;
   EFFECTS: porte_c.lock(id)
            AND id = 1
            AND circuitLocked = 1
            AND zctpLocked = 1
            AND actpLocked = 1;

   lockCircuitSubGroup;
   COND: cSLocked = 0;
   EFFECTS: porte_cs.lock(id)
            AND id = 1
            AND cSLocked = 1
            AND cSTPZLocked = 1
            AND cSTPALocked = 1;

   instanciateCircuit;
   COND: circuitInstalled = 0;
   EFFECTS: porte_c.create_circuit(id,ep,ep1)
          AND circuitLocked = 1
          AND zctpLocked = 1
          AND actpLocked = 1
          AND id =1
          AND ep =1
          AND ep1 = 1;

   END_RULE_SYSTEM OPERATOR;
```

```
   unlockZCTP;
   COND: actpLocked = 1;
   EFFECTS: porte_a.unlock(id)
         AND id = 2
            AND zctpLocked = 0;

   lockZsubgroupTP;
   COND: cSTPZLocked = 0;
   EFFECTS: cSTPZLocked = 1
            AND id = 2
            AND porte_cstp.lock(id);

   unlockZsubgroupTP;
   COND: cSTPZLocked = 1;
   EFFECTS: porte_cstp.unlock(id)
    AND id = 2
            AND cSTPZLocked = 0;

   unlockCircuit;
   COND: circuitLocked = 1;
   EFFECTS: porte_c.unlock(id)
            AND id = 1
            AND circuitLocked = 0
            AND zctpLocked = 0
            AND actpLocked = 0;

   unlockCircuitSubGroup;
   COND: cSLocked = 1;
   EFFECTS: porte_cs.unlock(id)
         AND id = 1
            AND cSLocked = 0
            AND cSTPZLocked = 0
            AND cSTPALocked = 0;

   termintateCircuit;
    COND: circuitInstalled = 1;
    EFFECTS: porte_c.delete_circuit(id)
      AND id =1;
```

6.2.3 L'équipement

```
RULE_SYSTEM EQUIPMENT(  mgmt            : PORTE;
                        porte_t_ttp     : PORTE;
                        identifier   : INTEGER;
                        ttpIdf          : INTEGER);
STATE

  eqId                          : INTEGER INITIALLY identifier;
  administrativeState           : AdministrativeState INITIALLY unlocked;
  operationalState              : OperationalState INITIALLY enabled;
  availabilityStatus            : INTEGER INITIALLY 0;

DECLARATIONS

  id                            : INTEGER;
```

```
RULES

lock;
COND: mgmt.lock(id)
     AND identifier = id;
EFFECTS: administrativeState = locked
        AND id = ttpIdf
        AND porte_t_ttp.disable(id);

unlock;
COND: mgmt.unlock(id)
     AND identifier = id;
EFFECTS: administrativeState = unlocked
        AND id = ttpIdf
        AND porte_t_ttp.enable(id);

END_RULE_SYSTEM EQUIPMENT;
```

6.2.4 Le point de terminaison physique

```
RULE_SYSTEM TTP( identifier  : INTEGER;
                 ctpIdf          : INTEGER;
                 porte           : PORTE;
                 porte_t_ctp     : PORTE;
                 porte_f_eq      : PORTE);

STATE

  idf                : INTEGER INITIALLY identifier;
  ctpId              : INTEGER INITIALLY ctpIdf;
  administrativeState : AdministrativeState INITIALLY unlocked;
  operationalState    : OperationalState INITIALLY enabled;
  availabilityStatus  : INTEGER INITIALLY 0;

DECLARATIONS

  id   : INTEGER;

RULES

enable;
COND: porte_f_eq.enable(id)
      AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus - 1
      AND id = ctpIdf
      AND porte_t_ctp.enable(id);

disable;
COND: porte_f_eq.disable(id)
      AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus + 1
      AND id = ctpIdf
      AND porte_t_ctp.disable(id);

lock;
COND: mgmt.lock(id)
      AND id = identifier;
EFFECTS: administrativeState = locked
      AND id = ctpIdf
        AND porte_t_ctp.disable(id);

unlock;
COND: mgmt.unlock(id)
      AND id = identifier;
EFFECTS: administrativeState = unlocked
        AND id = ctpIdf
        AND porte_t_ctp.enable(id);

END_RULE_SYSTEM TTP;
```

6.2.5 Le point de terminaison de circuit

```
RULE_SYSTEM CTP( identifier   : INTEGER; // The Ctp idf
                 porte           : PORTE;  // Management Interface
                 porte_f_ttp     : PORTE;  // Relationship Interface to TTP
                 porte_f_cstp    : PORTE;
                 porte_t_circuit : PORTE;
                 porte_f_circuit : PORTE);

STATE

  administrativeState : AdministrativeState INITIALLY unlocked;
```

```
  operationalState    : OperationalState INITIALLY disabled;
  availabilityStatus  : INTEGER INITIALLY 1;

// Status and rule states

  attachedCircuit   : INTEGER INITIALLY 0;
  propagate_disable : INTEGER INITIALLY 0;
  propagate_enable  : INTEGER INITIALLY 0;

DECLARATIONS

  circuitId  : INTEGER;
  cid        : INTEGER;
  id         : INTEGER;
  id1        : INTEGER;
  ep         : INTEGER;
  ep1        : INTEGER;

RULES

CONTEXT attachment;

RULES

getdep;
COND: porte_f_circuit.get_dependency(id)
      AND identifier = id;
EFFECTS:id1 = identifier
      AND IF 'administrativeState = locked
          THEN cid = 'availabilityStatus + 1
          ELSE cid = 'availabilityStatus
          FI
      AND attachedCircuit = 1
      AND porte_t_circuit.receive_dependency(id1,cid);

setdep;
COND: porte_f_circuit.set_dependency(id,ep)
      AND identifier = id;
EFFECTS: availabilityStatus = 'availabilityStatus + ep;

removedep;
COND: porte_f_circuit.remove_dependency(id,ep)
      AND identifier = id;
EFFECTS: availabilityStatus = 'availabilityStatus - ep;

END_CONTEXT attachment;

   ttpenable;
   COND: porte_f_ttp.enable(id)
         AND id = identifier;
   EFFECTS:
     availabilityStatus = 'availabilityStatus - 1
           AND IF (availabilityStatus = 0)
               THEN operationalState = enabled
               ELSE TRUE
               FI
           AND IF 'attachedCircuit = 1
               THEN propagate_enable = 1
               ELSE propagate_enable = 0
               FI;

   issue_enable;
   COND: propagate_enable = 1;
   EFFECTS:  propagate_enable = 0
           AND porte_t_circuit.enable(id)
           AND id = identifier;

ttpdisable;
COND: porte_f_ttp.disable(id)
      AND id = identifier;
EFFECTS:
  availabilityStatus = 'availabilityStatus + 1
        AND IF (availabilityStatus = 1)
            THEN operationalState = disabled
            ELSE TRUE
            FI
        AND IF 'attachedCircuit = 1
            THEN propagate_disable = 1
            ELSE propagate_disable = 0
            FI;

issue_disable;
COND: propagate_disable = 1;
EFFECTS: propagate_disable = 0
      AND porte_t_circuit.disable(id)
      AND id = identifier;
```

```
adminLock;
COND: mgmt.lock(id)
    AND id = identifier;
EFFECTS: IF 'administrativeState = locked
         THEN TRUE
         ELSE administrativeState = locked
              AND IF 'attachedCircuit = 1
                  THEN propagate_disable = 1
                  ELSE propagate_disable =0
                  FI
         FI;

cstpenable;
COND: porte_f_cstp.enable(id)
              AND id = identifier;
EFFECTS:
  availabilityStatus = 'availabilityStatus - 1
         AND IF availabilityStatus = 0
             THEN operationalState = enabled
             ELSE TRUE
             FI
         AND IF 'attachedCircuit = 1
                       THEN propagate_enable = 1
                       ELSE propagate_enable = 0
                       FI;

lockProp;
COND: porte_f_circuit.lock(id)
               AND id = identifier;
EFFECTS: IF 'administrativeState = locked
         THEN TRUE
         ELSE administrativeState = locked
             AND propagate_disable = 1
         FI;


disable;
COND: porte_f_circuit.disable(id)
    AND id = identifier;
EFFECTS: operationalState = disabled
  AND availabilityStatus = 'availabilityStatus + 1;
```

```
adminUnlock;
COND: mgmt.unlock(id)
          AND id = identifier;
EFFECTS: IF 'administrativeState = unlocked
         THEN TRUE
         ELSE administrativeState = unlocked
                   AND IF 'attachedCircuit = 1
                       THEN propagate_enable = 1
                       ELSE propagate_enable = 0
                       FI
         FI;

cstpdisable;
COND: porte_f_cstp.disable(id)
              AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus + 1
         AND IF availabilityStatus = 1
             THEN operationalState = disabled
             ELSE TRUE
             FI
         AND IF 'attachedCircuit = 1
                       THEN propagate_disable = 1
                       ELSE propagate_disable = 0
             FI;


unlockProp;
COND: porte_f_circuit.unlock(id)
              AND id = identifier;
EFFECTS: IF 'administrativeState = unlocked
         THEN TRUE
         ELSE porte_t_circuit.enable(id1)
              AND id1 = identifier
      AND administrativeState = unlocked
         FI;

enable;
COND: porte_f_circuit.enable(id)
              AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus - 1
         AND IF availabilityStatus = 0
             THEN operationalState = enabled
             ELSE TRUE
             FI;

lockf;
COND: porte_f_circuit.lockf(id)
    AND id = identifier;
EFFECTS: administrativeState = locked
    AND availabilityStatus = 'availabilityStatus +1
    AND operationalState = disabled
    AND attachedCircuit = 0;
```

```
END_RULE_SYSTEM CTP;
```

6.2.6 Le point de terminaison de sous-groupe de circuits

```
RULE_SYSTEM CSTP( identifier  : INTEGER;
                  porte           : PORTE;
                  porte_t_ccs     : PORTE;
                  porte_f_ccs     : PORTE;
                  porte_t_ctp     : PORTE);
```

```
STATE

  administrativeState           : AdministrativeState INITIALLY unlocked;
  operationalState              : OperationalState INITIALLY enabled;
  availabilityStatus            : INTEGER INITIALLY 0;
  disableCtp                    : INTEGER INITIALLY 0;
  disableSubGroup               : INTEGER INITIALLY 0;
  enableCtp                     : INTEGER INITIALLY 0;
  enableSubGroup                : INTEGER INITIALLY 0;

DECLARATIONS

  id : INTEGER;

RULES

disableCtp_prop;
COND: disableCtp = 1;
EFFECTS: disableCtp = 0
        AND id = identifier
        AND porte_t_ctp.disable(id);

disableSubGroup_prop;
COND: disableSubGroup = 1;
EFFECTS: disableSubGroup = 0
        AND id = identifier
        AND porte_t_ccs.disable(id);

lock;
COND: mgmt.lock(id)
          AND id = identifier;
EFFECTS: IF 'administrativeState = locked
         THEN TRUE
         ELSE disableCtp = 1
     AND disableSubGroup = 1
             AND administrativeState = locked
         FI;

lock2;
COND: porte_f_ccs.lock(id)
        AND id = identifier;
EFFECTS: IF 'administrativeState = locked
         THEN TRUE
         ELSE disableCtp = 1
     AND disableSubGroup = 1
             AND administrativeState = locked
         FI;

enable;
COND: porte_f_ccs.enable(id)
        AND id = identifier;
EFFECTS:
  availabilityStatus = 'availabilityStatus - 1
         AND IF availabilityStatus = 0
             THEN operationalState = enabled
             ELSE TRUE
             FI;

END_RULE_SYSTEM CSTP;
```

```
enableCtp_prop;
COND: enableCtp = 1;
EFFECTS: enableCtp = 0
        AND id = identifier
        AND porte_t_ctp.enable(id);

enableSubGroup_prop;
COND: enableSubGroup = 1;
EFFECTS: enableSubGroup = 0
        AND id = identifier
        AND porte_t_ccs.enable(id);

unlock;
COND: mgmt.unlock(id)
        AND id = identifier;
EFFECTS: IF 'administrativeState = unlocked
         THEN TRUE
         ELSE enableSubGroup = 1
              AND enableCtp = 1
              AND administrativeState = unlocked
         FI;

unlock2;
COND: porte_f_ccs.unlock(id)
         AND id = identifier;
EFFECTS: IF 'administrativeState = unlocked
         THEN TRUE
         ELSE enableSubGroup = 1
              AND enableCtp = 1
              AND administrativeState = unlocked
         FI;

disable;
COND: porte_f_ccs.disable(id)
        AND id = identifier;
EFFECTS: operationalState = disabled
   AND availabilityStatus = 'availabilityStatus + 1;
```

6.2.7 Le circuit

```
RULE_SYSTEM CIRCUIT ( identifier : INTEGER;
                      porte         : PORTE;
                      porte_f_cs    : PORTE;
                      porte_f_ctp   : PORTE;
                      porte_t_ctp   : PORTE);

STATE

  aCtp                : INTEGER             INITIALLY 0;
  zCtp                : INTEGER             INITIALLY 0;
  administrativeState : AdministrativeState INITIALLY locked;
  operationalState    : OperationalState    INITIALLY enabled;
  availabilityStatus  : INTEGER             INITIALLY 0;

// Intermediate variables
  vz_to_lock    : INTEGER  INITIALLY 0;          va_to_lock    : INTEGER  INITIALLY 0;
  vz_to_lockf   : INTEGER  INITIALLY 0;          va_to_lockf   : INTEGER  INITIALLY 0;
  vz_to_disable : INTEGER  INITIALLY 0;          va_to_disable : INTEGER  INITIALLY 0;
  vz_to_unlock  : INTEGER  INITIALLY 0;          va_to_unlock  : INTEGER  INITIALLY 0;
  vz_to_enable : INTEGER  INITIALLY 0;           va_to_enable : INTEGER  INITIALLY 0;
  toz           : INTEGER  INITIALLY 1;          toa           : INTEGER  INITIALLY 1;
  getz          : INTEGER  INITIALLY 0;          geta          : INTEGER  INITIALLY 0;
  setz          : INTEGER  INITIALLY 0;          seta          : INTEGER  INITIALLY 0;
  initStep      : INTEGER  INITIALLY 0;          releaseStep   : INTEGER  INITIALLY 0;
  fromZ         : INTEGER  INITIALLY 0;          fromA         : INTEGER  INITIALLY 0;
  attached      : INTEGER  INITIALLY 0;
DECLARATIONS

  id  : INTEGER;
  ep  : INTEGER;
  ep1 : INTEGER;
  ept  : INTEGER;

RULES

CONTEXT initialize;
  COND: attached = 0;

RULES

create_circuit;
 COND: mgmt.create_circuit(id,ep,ep1);
 EFFECTS: aCtp = ep
        AND zCtp = ep1
        AND geta = 1
        AND getz =1;

getadep;
 COND: geta =1;
 EFFECTS: id = identifier
     AND ep = 'aCtp
     AND geta = 0
     AND initStep = 'initStep + 1
     AND porte_t_ctp.get_dependency(ep);

getzdep;
  COND: getz =1;
  EFFECTS: id = identifier
      AND ep = 'zCtp
      AND getz = 0
      AND initStep = 'initStep + 1
      AND porte_t_ctp.get_dependency(ep);
```

```
   receive_adep;
   COND: porte_f_ctp.receive_dependency(ept,ep1)
     AND ept = aCtp;
   EFFECTS:
    availabilityStatus = 'availabilityStatus + ep1 - 1
       AND fromA = ep1 - 1
       AND IF NOT (ep1  = 1)
           THEN toz = ep1
               AND initStep = 'initStep + 1
           ELSE initStep = 'initStep + 2
           FI;

   setaDep;
   COND: NOT(toa = 1)
     AND geta = 0;
   EFFECTS: porte_t_ctp.set_dependency(id,ep1)
        AND initStep = 'initStep + 1
        AND ep1 = 'toa - 1
        AND toa = 1
        AND id = 'aCtp;

   terminate_init;
   COND: initStep = 6;
   EFFECTS: administrativeState = locked
        AND vz_to_lock = 1
        AND va_to_lock = 1
        AND vz_to_disable = 1
        AND va_to_disable = 1
        AND releaseStep = 0
        AND attached =1;

END_CONTEXT initialize;

CONTEXT release;
  COND: attached = 1;

RULES

   delete_circuit;
   COND: mgmt.delete_circuit(id)
     AND id = identifier;
   EFFECTS: seta = 1
      AND releaseStep = 1
      AND setz = 1
      AND vz_to_lockf = 1
      AND va_to_lockf = 1;

   setaDep1;
   COND:seta = 1;
   EFFECTS: porte_t_ctp.remove_dependency(id,ep1)
        AND releaseStep = 'releaseStep + 1
        AND ep1 = 'fromZ
        AND seta = 0
        AND id = 'aCtp;

END_CONTEXT release;

CONTEXT normal_use;
 COND: attached =1 AND initStep = 6 AND releaseStep = 0;

RULES
```

```
   receive_zdep;
   COND: porte_f_ctp.receive_dependency(ept,ep1)
     AND ept = zCtp;
   EFFECTS:
    availabilityStatus = 'availabilityStatus + ep1 - 1
       AND fromZ = ep1 - 1
       AND IF NOT (ep1  = 1)
           THEN toa = ep1
              AND initStep = 'initStep + 1
           ELSE initStep = 'initStep + 2
          FI;

   setzDep;
   COND: NOT(toz = 1)
      AND getz = 0;
   EFFECTS: porte_t_ctp.set_dependency(id,ep1)
        AND initStep = 'initStep + 1
        AND ep1 = 'toz - 1
        AND toz = 1
        AND id = 'zCtp;
```

```
   terminate_release_p;
   COND: releaseStep = 3;
   EFFECTS: attached = 0
        AND initStep = 0;
```

```
   setzDep1;
   COND: setz = 1;
   EFFECTS: porte_t_ctp.remove_dependency(id,ep1)
        AND releaseStep = 'releaseStep + 1
        AND ep1 = 'fromA
        AND setz = 0
        AND id = 'zCtp;
```

```
glock;
COND: mgmt.lock(id)
 AND attached =1
 AND identifier = id;
EFFECTS: administrativeState = locked
 AND vz_to_lock = 1
 AND va_to_lock = 1
 AND va_to_disable = 1
 AND vz_to_disable = 1;
```

```
unlock;
COND: mgmt.unlock(id)
  AND attached =1
  AND identifier = id;
EFFECTS: administrativeState = unlocked
  AND vz_to_unlock = 1
  AND va_to_unlock = 1
  AND vz_to_enable = 1
  AND va_to_enable = 1;
```

```
enable;
COND: porte_f_ctp.enable(ep);
EFFECTS: availabilityStatus = 'availabilityStatus - 1
     AND IF availabilityStatus = 0
         THEN operationalState = enabled
 ELSE TRUE
         FI
     AND IF ep = 'aCtp
         THEN ep1 = 'zCtp
            AND fromA = 'fromA - 1
         ELSE ep1 = 'aCtp
            AND fromZ = 'fromZ - 1
         FI
            AND porte_t_ctp.enable(ep1);
```

```
disable;
COND: porte_f_ctp.disable(ep);
EFFECTS: operationalState = disabled
     AND availabilityStatus = 'availabilityStatus + 1
     AND IF ep = 'aCtp
         THEN ep1 = 'zCtp
           AND fromA = 'fromA + 1
         ELSE ep1 = 'aCtp
           AND fromZ = 'fromZ + 1
         FI
    AND porte_t_ctp.disable(ep1);
```

```
csenable;
COND: porte_f_cs.enable(id)
  AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus - 1
     AND IF availabilityStatus = 0
         THEN operationalState = enabled
         ELSE TRUE
         FI;
```

```
csdisable;
COND: porte_f_cs.disable(id)
  AND id = identifier;
EFFECTS: availabilityStatus = 'availabilityStatus + 1;
```

```
CONTEXT normal_use;
 COND: attached =1 AND initStep = 6 AND releaseStep = 0;

RULES

END_CONTEXT normal_use;

CONTEXT propagation;

RULES
```

```
az_lock_propf;
COND: va_to_lockf = 1;
EFFECTS: va_to_lockf = 0
     AND porte_t_ctp.lockf(ep)
     AND ep = 'aCtp;
```

```
vz_lock_propf;
COND: vz_to_lockf = 1;
EFFECTS: vz_to_lockf = 0
     AND porte_t_ctp.lockf(ep)
     AND ep = 'zCtp;
```

```
az_lock_prop;
COND: va_to_lock = 1;
EFFECTS: va_to_lock = 0
     AND porte_t_ctp.lock(ep)
     AND ep = 'aCtp;
```

```
vz_lock_prop;
COND: vz_to_lock = 1;
EFFECTS: vz_to_lock = 0
     AND porte_t_ctp.lock(ep)
     AND ep = 'zCtp;
```

```
   az_disable_prop;
   COND: va_to_disable = 1;
   EFFECTS: va_to_disable = 0
        AND porte_t_ctp.disable(ep)
        AND ep = 'aCtp;

   az_unlock_prop;
   COND: va_to_unlock = 1;
   EFFECTS: va_to_unlock = 0
        AND porte_t_ctp.unlock(ep)
        AND ep = 'aCtp;

   az_enable_prop;
   COND: va_to_enable = 1;
   EFFECTS: va_to_enable = 0
        AND porte_t_ctp.enable(ep)
        AND ep = 'aCtp;
```

```
   vz_disable_prop;
   COND: vz_to_disable = 1;
   EFFECTS: vz_to_disable = 0
        AND porte_t_ctp.disable(ep)
        AND ep = 'zCtp;

   vz_unlock_prop;
   COND: vz_to_unlock = 1;
   EFFECTS: vz_to_unlock = 0
       AND porte_t_ctp.unlock(ep)
       AND ep = 'zCtp;

   vz_enable_prop;
   COND: vz_to_enable = 1;
   EFFECTS: vz_to_enable = 0
       AND porte_t_ctp.enable(ep)
       AND ep = 'zCtp;
```

```
END_CONTEXT propagation;

END_RULE_SYSTEM CIRCUIT;
```

6.2.8 Le sous-groupe de circuits

```
RULE_SYSTEM SUBGROUP ( idf           : INTEGER;
porte         : PORTE;
  porte_t_cs    : PORTE;
  porte_f_cstp : PORTE;
  porte_t_cstp : PORTE);

STATE

  csIdf                        : INTEGER INITIALLY   idf;
  aCsTP                        : INTEGER INITIALLY   1;
  zCsTP                        : INTEGER INITIALLY   2;
  administrativeState          : AdministrativeState INITIALLY unlocked;
  operationalState             : OperationalState    INITIALLY enabled;
  availabilityStatus           : INTEGER             INITIALLY 0;
  vz_to_lock  : INTEGER            INITIALLY 0;
  va_to_lock  : INTEGER            INITIALLY 0;
  vz_to_disable  : INTEGER            INITIALLY 0;
  va_to_disable  : INTEGER            INITIALLY 0;
  vz_to_unlock  : INTEGER            INITIALLY 0;
  va_to_unlock  : INTEGER            INITIALLY 0;
  vz_to_enable  : INTEGER            INITIALLY 0;
  va_to_enable  : INTEGER            INITIALLY 0;
  circuit_to_enable  : INTEGER            INITIALLY 0;
  circuit_to_disable  : INTEGER            INITIALLY 0;

DECLARATIONS

  id    : INTEGER;

RULES
```

```
lock;
COND: mgmt.lock(id)
     AND id = 1;
EFFECTS: administrativeState = locked
AND vz_to_lock = 1
AND va_to_lock = 1
AND va_to_disable = 1
AND circuit_to_disable = 1
AND vz_to_disable = 1;

vz_lock_prop;
COND: vz_to_lock = 1;
EFFECTS: vz_to_lock = 0
     AND id = 'zCsTP
        AND porte_t_cstp.lock(id);

vz_disable_prop;
COND: vz_to_disable = 1;
EFFECTS: vz_to_disable = 0
     AND id = 'zCsTP
        AND porte_t_cstp.disable(id);

vz_unlock_prop;
COND: vz_to_unlock = 1;
EFFECTS: vz_to_unlock = 0
     AND id = 'zCsTP
        AND porte_t_cstp.unlock(id);

vz_enable_prop;
COND: vz_to_enable = 1;
EFFECTS: vz_to_enable = 0
     AND id = 'zCsTP
        AND porte_t_cstp.enable(id);

circuit_disable_prop;
COND: circuit_to_disable = 1;
EFFECTS: circuit_to_disable = 0
     AND id = 1
        AND porte_t_cs.disable(id);

zdisable;
COND: porte_f_cstp.disable(id)
         AND id = 2;
EFFECTS: operationalState = disabled
         AND availabilityStatus = 'availabilityStatus + 1
         AND id = 1
         AND porte_t_cstp.disable(id);

zenable;
COND: porte_f_cstp.enable(id)
         AND id = 2;
EFFECTS: availabilityStatus = 'availabilityStatus - 1
         AND IF availabilityStatus = 0
             THEN operationalState = enabled
             FI
         AND id = 1
         AND porte_t_cstp.enable(id);

END_RULE_SYSTEM SUBGROUP;
```

```
unlock;
COND: mgmt.unlock(id)
     AND id = 1;
EFFECTS: administrativeState = unlocked
AND vz_to_unlock = 1
AND va_to_unlock = 1
AND vz_to_enable = 1
AND circuit_to_enable = 1
AND va_to_enable = 1;

az_lock_prop;
COND: va_to_lock = 1;
EFFECTS: va_to_lock = 0
     AND id = 'aCsTP
        AND porte_t_cstp.lock(id);

az_disable_prop;
COND: va_to_disable = 1;
EFFECTS: va_to_disable = 0
     AND id = 'aCsTP
        AND porte_t_cstp.disable(id);

az_unlock_prop;
COND: va_to_unlock = 1;
EFFECTS: va_to_unlock = 0
     AND id = 'aCsTP
        AND porte_t_cstp.unlock(id);

az_enable_prop;
COND: va_to_enable = 1;
EFFECTS: va_to_enable = 0
     AND id = 'aCsTP
        AND porte_t_cstp.enable(id);

circuit_enable_prop;
COND: circuit_to_enable = 1;
EFFECTS: circuit_to_enable = 0
     AND id = 1
        AND porte_t_cs.enable(id);

adisable;
COND: porte_f_cstp.disable(id)
    AND id = 1;
EFFECTS: operationalState = disabled
      AND availabilityStatus = 'availabilityStatus + 1
     AND id = 2
         AND porte_t_cstp.disable(id);

aenable;
COND: porte_f_cstp.enable(id)
     AND id = 1;
EFFECTS: availabilityStatus = 'availabilityStatus - 1
         AND IF availabilityStatus = 0
             THEN operationalState = enabled
             FI
         AND id = 2
         AND porte_t_cstp.enable(id);
```

6.2.9 Une configuration de base

```
RULE_SYSTEM INIT();

STATE

csTP            : PORTE;
ctp             : PORTE;
eqMgmt          : PORTE;
eqttp           : PORTE;
ttpMgmt         : PORTE;
ttp             : PORTE;
cstpctp         : PORTE;
ctp_vc          : PORTE;
c_vctp          : PORTE;
circuit         : PORTE;
cs              : PORTE;
ccsv            : PORTE;
vccs            : PORTE;

eqAMO       : EQUIPMENT(eqMgmt, eqttp, 1, 1);
eqZMO       : EQUIPMENT(eqMgmt, eqttp, 2, 2);
aTTPMO      : TTP(1,1,ttpMgmt,ttp, eqttp);
zTTPMO      : TTP(2,2,ttpMgmt,ttp, eqttp);
aCtpMO      : CTP(1, ctp, ttp, cstpctp, ctp_vc, c_vctp);
zCtpMO      : CTP(2, ctp, ttp, cstpctp, ctp_vc, c_vctp);
circuitMO   : CIRCUIT(1, circuit, cs, ctp_vc, c_vctp);
aCsTpMO     : CSTP(1,csTP,vccs,ccsv,cstpctp);
zCsTpMO     : CSTP(2,csTP,vccs,ccsv,cstpctp);
subgroupMO  : SUBGROUP(1,cs, cs, vccs, ccsv);
operatorMO  : OPERATOR(eqMgmt, ttpMgmt, ctp, circuit, csTP, cs);

END_RULE_SYSTEM INIT;
```

Unit´e de recherche INRIA Lorraine, Technopˆole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhˆone-Alpes, 46 avenue F´elix Viallet, 38031 GRENOBLE Cedex 1
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399