Validation du modèle de gestion d'interconnexion de commutateurs à l'aide de systèmes de transitions étiquetées

Emmanuel Nataf, Olivier Festor, André Schaff

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Validation du modèle de gestion d'interconnexion de commutateurs à l'aide de systèmes de transitions étiquetées

Emmanuel Nataf, Olivier Festor, André Schaff

N° 2769

15 janvier 1996

PROGRAMME 1

ISSN 0249-6399

Validation du modèle de gestion d'interconnexion de commutateurs à l'aide de systèmes de transitions étiquetées

Emmanuel Nataf, Olivier Festor, André Schaff

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

Rapport de recherche n° 2769 — 15 janvier 1996 — 25 pages

Résumé :

Dans le cadre de la construction de modèles d'information pour la gestion de réseaux, la spécification des comportements devient de plus en plus complexe. En particulier, les dépendances entre objets gérés nécessitent une spécification détaillée bien souvent omise dans les catalogues d'objets disponibles à ce jour.

Nous présentons dans ce rapport la validation entreprise sur les dépendances comportementales entre objets pour la gestion des interconnexions entre commutateurs. La validation entreprise repose sur une modélisation sous forme de systèmes de transitions étiquetées des interactions entre ces objets ainsi que sur l'utilisation du logiciel `MEC` pour la vérification de propriétés sur cette modélisation.

Mots-clé : Gestion de réseaux, OSI, Systèmes de transitions étiquetées, Validation

(Abstract: pto)

Unité de recherche INRIA Lorraine
Technôpole 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

Validation of the Switch Interconnection Management Model with Labelled Transition Systems

Abstract:

Specifying behaviors within information models for network management becomes more and more complex. Especially behavioral dependencies between Managed Objects requires a detailled and precise specification often neglegted in the information models.

In this report we present, the verification undertaken on the behavioral dependencies between Managed Objects representing switch interconnection ressources. The verification is build on a specification of the objects and their dependencies through Labelled Transition Systems and the use of the `MEC` software for property verification.

Key-words: Network Management, OSI, Labelled Transition System, Validation

1 Introduction

Les catalogues d'objets gérés définis pour la modélisation des ressources logiques et physiques des réseaux croissent régulièrement en complexité et en nombre d'objets. La spécification du comportement de ces derniers est cruciale à la fois pour les concepteurs d'agents et les applications qui exploitent les informations contenues dans ces objets.

Dans les catalogues existants, ces comportements et notamment les dépendances entre les différents objets sont principalement décrits de manière informelle et sont bien souvent incomplets. Or, fournir un modèle de l'information nécessite des spécifications précises et correctes. Ceci requiert, vu la complexité des modèles, d'une part la mise à disposition de langages et méthodes de spécification puissants et, d'autre part la vérification des spécifications ainsi établies.

Nous illustrons ici l'utilisation du logiciel `MEC` [Arnold 89] développé au LABRI, pour valider un sous-ensemble des dépendances de comportement issues d'un catalogue d'objets gérés qui modélise la gestion de la configuration de l'interconnexion entre commutateurs [NMF 94]. Ce catalogue est édité par le Network Management Forum. Nous avons retenu ce modèle de l'information car il comporte, d'une part des dépendances entre objets gérés clairement identifiées mais ne spécifie pas, d'après une première analyse empirique, de manière exhaustive et complète les comportements associés.

La démarche entreprise dans cet article consiste à reprendre le modèle, en identifier les incomplétudes et les incohérences puis de reprendre les règles qui spécifient le comportement, de les modifier et de les affiner et enfin, d'en construire une spécification formelle et la valider.

Cette étude est organisée comme suit. La section 2 présente le modèle de l'information à valider. La section 3 détaille les spécifications sous forme de systèmes de transitions étiquetées et de vecteurs de synchonisation du modèle ainsi que les vérifications entreprises. La section 4 résume les résultats obtenus lors de la validation. La section 5 résume le travail réalisé dans cette étude et dresse un bilan sur l'adéquation de l'outil `MEC` aux besoins de validation des modèles de l'information en gestion de réseaux. Une conclusion sur ce travail est donnée en 5.

2 Le modèle de gestion de la configuration d'interconnexion des commutateurs

Les objets gérés modélisés dans le document du NMF mettent à disposition une interface de gestion d'un réseau de commutateurs (RTS, RNIS,...). Le niveau d'abstraction choisi (cf. fig. 1) est celui du niveau réseau; entre le niveau élément et le niveau service (suivant la terminologie du TMN [CCITT.M.3010 92, CCITT.M.3100 92]). L'interface de niveau élément est étendue avec la représentation des connexions entre ces derniers. L'interface de réseau obtenue comporte les fonctions de base pour la spécification de services, par exemple l'établissement d'un réseau privé virtuel (VPN : *Virtual Private Network*). Le document se restreint aux connexions logiques entre des éléments d'un type particulier, les commutateurs (ou *switchs*). Cette abstraction est indépendante de tout type de technologie (de traitement et de transmission d'informations). Le modèle peut donc être instancié sur des configurations différentes ou être spécialisé sur une configuration précise.

2.1 La spécification des ressources

La méthode de modélisation adoptée est celle préconisée par le modèle de référence de gestion OSI de l'ISO [ISO-7498.4 89]. Les ressources sont abstraites à l'aide de classes d'objets décrites avec la syntaxe GDMO [1] [ISO-10165.5 92]. La figure 2 illustre une configuration du modèle. Dans cet exemple, un réseau (classe `network`) est composé de deux élément gérés (classe `managedElement`) qui sont, dans le contexte du document, les commutateurs eux même. Un élément géré est constitué de plusieurs équipements (classe `equipment`). Les éléments gérés établissent entre eux un sous-groupe de circuits (classe `circuitSubGroup`). Dans notre exemple, un tel sous-groupe est composé de 3 circuits (classe `interExchangeCircuit`). Ces circuits sont sensés avoir des propriétés communes (bande passante,...). Chaque circuit est terminé par deux points de terminaison de circuit (classe `circuitXtp`). Le sous-groupe de circuits est quant à lui, terminé par deux sous-groupes de points de terminaison (classe `circuitSubGroupTerminationPoint`) au sein des éléments gérés. Les points de terminaison physiques (classe `trailTerminationPointBidirectional`) sont associés à des équipements (classe `equipment`), ils représentent le matériel et logiciel réalisant le transfert entre les éléments gérés.

1. GDMO : Guideline for the Definition of Managed Object

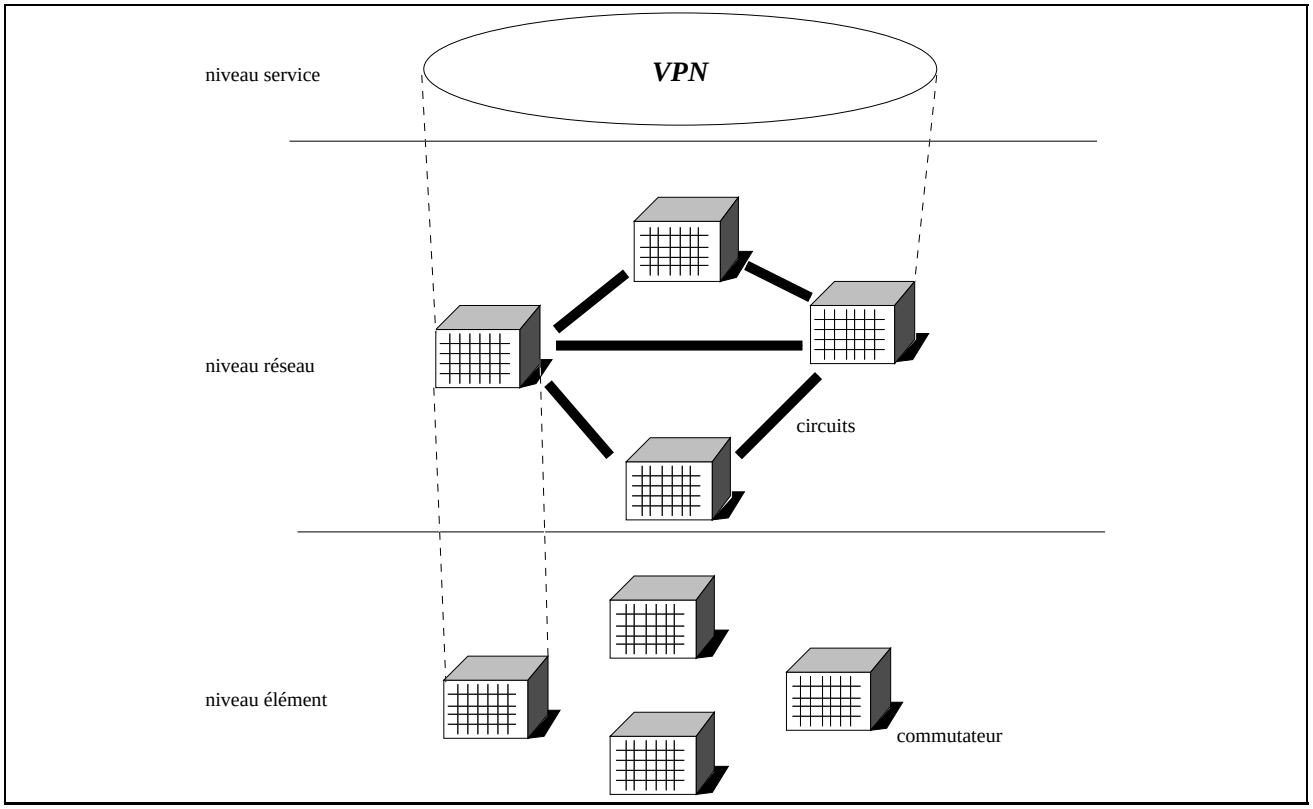

FIG. 1 – *Les niveaux d'abstractions*

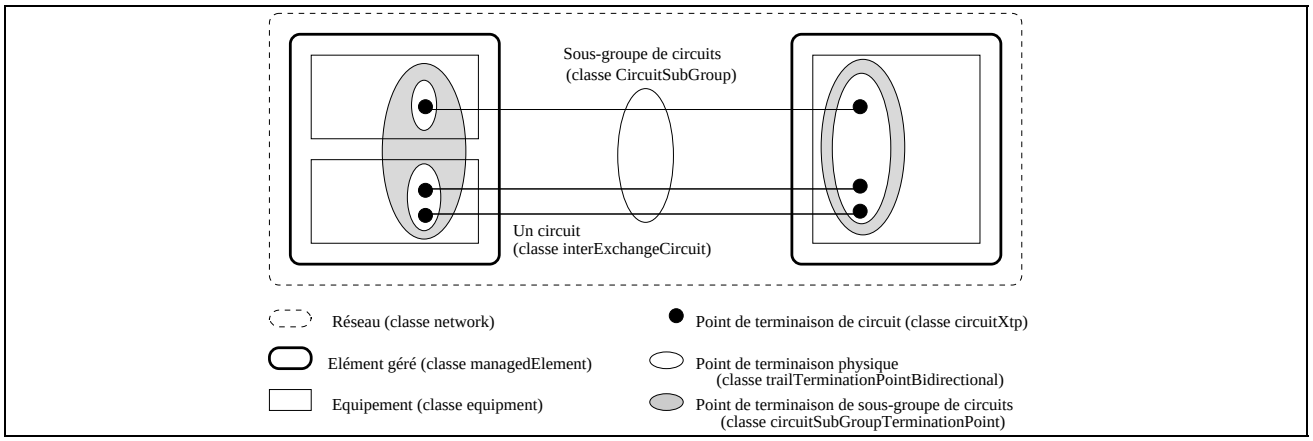

FIG. 2 – *Les composants du modèle*

Chaque élément d'une de ces classes exhibe un comportement propre. Ce comportement est modélisé par les actions autorisées sur un objet ainsi que par l'état de l'objet, c'est à dire le tuple des valeurs de ses attributs. Tout objet comporte les attributs suivants:

- un état administratif (`administrativeState`), dont les valeurs possibles sont `locked` ou `unlocked`. Cet attribut peut être à tout moment modifié par une action du responsable de gestion sur l'objet. La valeur de cet attribut peut dépendre de l'état d'autres objets.
- un état opérationnel (`operationalState`) pouvant prendre les valeurs `enabled` ou `disable`. Cet attribut ne peut pas être modifié par l'opérateur. Sa valeur reflète l'état de la ressource que l'objet modélise. Sa valeur peut dépendre de l'état d'autres objets.
- un statut de disponibilité (`availabilityStatus`), qui est un multi-ensemble pouvant contenir les valeurs `offline` et `dependency`. Cet attribut a pour but de préciser les raisons du blocage ou de l'indisponibi-

lité de la ressource. C'est un multi-ensemble car on verra plus loin que plusieurs dépendances peuvent contraindre un même objet. De plus, l'utilisation de GDMO pour les classes d'objets entraîne celle de ASN.1 [2] [ISO-8824 90] pour le type des attributs; or ASN.1 ne permet pas la description d'un ensemble au sens mathématique du terme.

Tout objet du modèle comporte chacun de ces attributs. La section suivante présente une description des dépendances identifiées entre ces objets.

2.2 Relations entre les MOs

Les objets sont contenus virtuellement dans une même MIB [3]. La répartition physique des commutateurs entraîne une distribution des objets gérés, le document [NMF 94] nous propose une vue rendant transparentes les communications (de gestion) nécessaires au maintien de la cohérence entre les objets. La figure 3 montre la structure de la MIB (classes d'objets, dépendance de contenance en trait plein), ainsi que deux autres types de relations (en pointillés) détaillées dans les sections suivantes. L'exemple de la figure 2 est une instantiation de cette structure, où le réseau (`network`) contient deux `managedElement` et un `circuitSubGroup`, ce dernier reliant deux `circuitSubGroupTerminationPoint`. Avant de donner les comportements associés aux relations, signalons que les relations de contenances sont spécifiées avec des formulaire de corrélation de noms (NAME BINDING dans [ISO-10165.5 92]) et que les autres relations le sont avec des attributs de relations comme dans [ISO-10164.3 92].

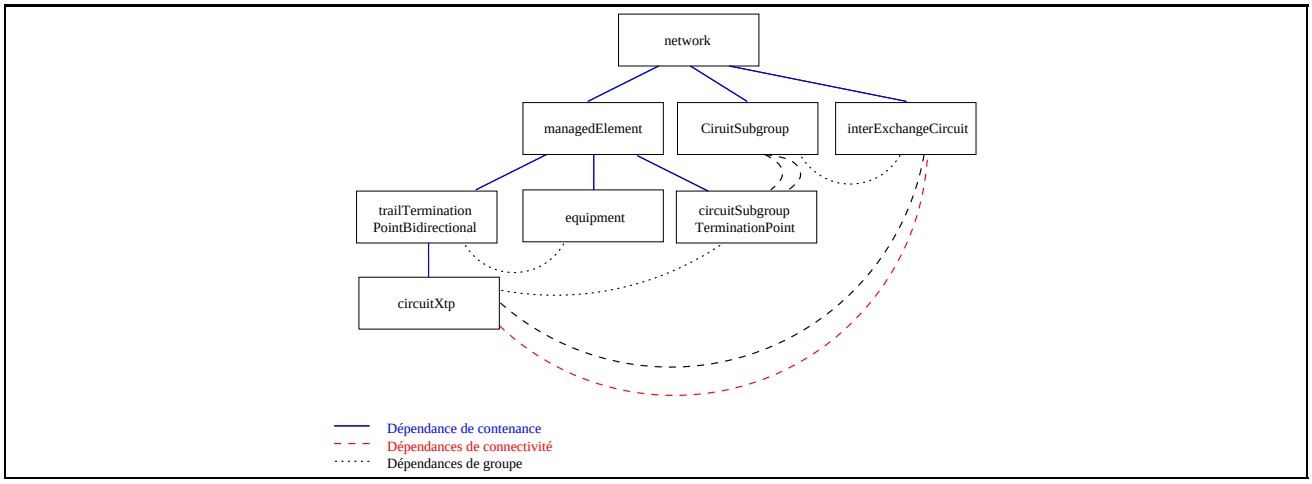

FIG. 3 – *Les dépendances comportementales identifiées*

A chaque dépendance (ou relation), des contraintes comportementales sont associées. Celles-ci sont définies dans les sections suivantes.

2.2.1 Dépendance de Connectivité

La relation de connectivité lie les ressources qui modélisent des liens entre éléments gérés et les points de terminaison correspondants. On retrouve donc cette relation entre deux `circuitXtp` et un `interExchangeCircuit` ainsi qu'entre deux `circuitSubGroupTerminationPoint` et un `circuitSubGroup`. D'une manière globale, on parlera de points de terminaison et de connectivité.

Pour les dépendances comportementales, le document stipule que :

- une mise à la valeur `locked` d'un objet point de terminaison doit entraîner la mise à la valeur `disabled` de l'objet de connectivité et de l'objet point de terminaison associé.
- une mise à la valeur `locked` d'un objet de connectivité doit entraîner la mise à la valeur `locked` des deux points de terminaison, ainsi que leur mise à la valeur `disabled`.

2. ASN.1 : Abstract Syntax Number One
3. MIB : Management Information Base

– une mise à la valeur `unlocked` de ce même objet entraînera la mise à la valeur `unlocked` et `enabled` des objets points de terminaison, ainsi que la mise (ou le maintien) de la valeur `enabled` de l'objet de connectivité.

2.2.2 Dépendance de Contenance

Les objets liés par la relation de contenance sont ceux qui sont instanciés les uns par rapport aux autres dans une Base d'Information de Gestion au travers de corrélations de noms.

Les règles de comportement associées précisent qu'une modification de la valeur de l'attribut `operationalState` dans un objet supérieur doit entraîner la même modification dans les objets subordonnés. De plus, si un objet subordonné est un point de terminaison (cf 2.2.1) entrant dans une relation de connectivité, alors les autres objets reliés (connectivité et point de terminaison opposé) voient également leur état opérationnel modifié.

2.2.3 Dépendance de groupe

La relation de dépendance de groupe complète les deux relations précédentes. Elle comporte deux rôles, un rôle fédérateur et un rôle fédéré. Elle relie explicitement les classes selon le tableau suivant défini dans [NMF 94].

Féderateur	Fédéré
Point de terminaison de groupe de circuit (`circuitSubGroupTerminationPoint`)	Point de terminaison de circuit (`circuitXtp`)
Sous-groupe de circuits (`circuitSubGroup`)	Circuit (`interExchangeCircuit`)
Equipement (`equipment`)	Point de terminaison physique (`trailTerminationPointBidirectional`)

Les règles comportementales associées sont:

– la mise à la valeur `locked` d'un objet fédérateur entraine une mise à la valeur `disabled` de l'objet fédéré correspondant.

– la mise à la valeur `unlocked` d'un objet fédérateur entraîne la mise à la valeur `enabled` de l'objet fédéré correpondant.

2.2.4 Règles supplémentaires

Deux règles supplémentaires définissent les valeurs que doit prendre l'attribut `availabilityStatus`, selon les modifications des attributs `administrativeState` ou `operationalState`. Ainsi, pour l'état opérationnel, son passage à `disabled`, dû à la présence de l'objet dans une des relation ci-dessus, lui rajoute la valeur `dependency`, et sa mise à la valeur `enabled` lui enlève cette valeur. Enfin, pour tout objet, la mise à la valeur `locked` entraîne l'ajout de la valeur `offline` dans le statut de disponibilité, valeur qui sera enlevée lors du déblocage administratif (passage à `unlocked`).

Il est à noter que ces règles peuvent à priori s'appliquer de manière récursive lors d'une transition. Les seuls stimulis pouvant provenir de l'extérieur, sont des actions de blocage administratif des différentes ressources ainsi que des blocages opérationnels des ressources liées à des éléments physiques du réseau (équipements, points de terminaison physiques, ...).

2.3 Les incomplétudes et inconsistances des règles

Les règles de comportement présentées ci-dessus sont incomplètes car elles ne décrivent pas l'ensemble des transitions possibles. Notamment le déblocage administratif (`unlock`) de certains éléments comme les points de terminaison de circuits n'est pas décrit explicitement. Une approche réaliste consiste à prendre le comportement inverse à celui décrit pour le blocage. C'est ce que nous avons fait.

Le second problème dans ces règles de comportement est celui lié à l'interprétation des transitions et de la propagation des dépendances associées. En effet, la lecture stricte de ces règles implique la propagation des dépendances uniquement lors des transitions des états opérationnels (de `enabled` vers `disabled` et inversement) ou de l'état administratif (de `unlocked` vers `locked` et inversement).

Sur la base de cette interprétation, nous avons construit un modèle sous forme de systèmes de transitions étiquetées et entrepris une série de validation à l'aide de `MEC`. Très vite, c'est-à-dire lors de la modélisation,

nous avons identifié des états incohérents dans le système. La validation `MEC` nous a confirmé ces soupçons. Un exemple d'état incohérent identifié est le suivant: un point de terminaison est opérant (état opérationnel à `enabled`) et l'équipement associé par l'intermédiaire d'un point de terminaison physique est bloqué (état administratif à `locked`). Pour atteindre un tel état global, il suffit de bloquer l'équipement, ce qui entraine l'inopérabilité du point de terminaison, puis de bloquer et de débloquer l'objet de connectivité associé à ce dernier, ce qui le rendra à nouveau opérationnel (avec un équipement toujours bloqué!).

A partir de ce constat, nous avons décidé de reformuler les règles existantes et de valider le nouveau comportement. Pour illustrer l'utilisation de `MEC` dans ce but, nous présentons dans la suite le nouveau modèle établi.

3 La modélisation en MEC et la validation

Dans un premier temps, nous avons élaboré un modèle comportant une configuration complète du système. Cette configuration n'a pu être validée en une seule partie en raison de sa trop grande consommation de mémoire sous `MEC`.

Après identification des points critiques du modèle, nous l'avons divisé en deux spécifications distinctes sur lesquelles nous avons vérifié les propriétés indiquées comme caractéristiques du modèle. Après une explication sur les modélisations de bases choisies, nous montrons les deux configurations testées et les résultats obtenus.

3.1 Les composants de base des modèles

Tout objet d'un modèle est composé de deux systèmes de transitions étiquetées. L'un représente l'état administratif de l'objet, l'autre l'état opérationnel conjugé aux dépendances éventuelles. Nous n'avons pas modélisé la caractéristique `offline` qu'un objet affiche lorsqu'il est bloqué administrativement car cette caractéristique est duale à l'état administratif et n'implique aucune autre dépendance.

L'état administratif est modélisé par un système de transitions simple comportant deux états qui sont `locked` et `unlocked` (cf fig. 4 pour l'automate et la formalisation en découlant).

```
transition_system AdministrativeState;

unlocked  |- e      -> unlocked,
             lock   -> locked;
locked    |- e      -> locked,
             unlock -> unlocked;

<initial = {unlocked}>.
```

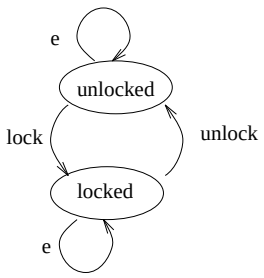

FIG. 4 – *Le système de transitions étiquetées de l'état administratif*

Le second système de transition modélise l'état opérationnel lié aux dépendances. Nous avons volontairement regroupé ces deux aspects car l'existence d'une ou plusieurs dépendances implique que l'objet concerné soit à l'état opérationnel `disabled`. Ce système de transitions peut donc exhiber la valeur `enabled` ou `disabledDep*` où `*` représente le nombre de dépendances affectées à l'objet à un moment donné. Cette valeur peut être supérieure à 1 car l'attribut `availabilityStatus` est de type multi-ensemble et des dépendances peuvent provenir de différents objets associés.

Il n'est pas possible de modéliser un tel attribut de manière exhaustive avec un système de transitions fini : il faudrait, partant d'un état initial, permettre un nombre de fois inconnu l'ajout d'une valeur et assurer que le retrait de cette valeur le même nombre de fois ramène l'automate à l'état initial. Nous l'avons défini au travers d'une modélisation intermédiaire en limitant le nombre de dépendances. Une étude des configurations possibles de relations montre que la valeur `dependency` ne peut être ajoutée qu'un nombre fini de fois, ici 10. Pour voir cela, nous avons cherché le nombre maximum de relations auquel un objet peut participer ainsi que les relations indirectes dans lesquelles des propagations peuvent se produire. Si l'on considère l'ensemble de notre modèle on arrive à un degré maximal de dépendance à 10. En effet au travers des relations, il existe le circuit (`interExchangeCircuit`) qui est potentiellement en relation (directement ou indirectement) avec tous les autres objets du modèle. Il y à 9 objets pouvant y induire une dépendance, plus lui-même, soit donc un total de 10.

```
transition_system OperationalStateDependency;

enabled    |-e             -> enabled,
             disablepDep   -> disabledDep,
             disableppDep  -> disabledDep2,
             disablepppDep -> disabledDep3;

disabledDep  |-enablemDep    -> enabled,
               disablepDep   -> disabledDep2,
               disableppDep  -> disabledDep3,
               disablepppDep -> disabledDep4,
               e             -> disabledDep;
...

disabledD10  |-enablemDep    -> disabledDep9,
               enablemmDep   -> disabledDep8,
               enablemmmDep  -> disabledDep7,
               e             -> disabledD10;
<initial = {enabled}>.
```

FIG. 5 – *Le système de transition de l'état opérationnel incluant les dépendances*

L'étude détaillée des propagations montre qu'au maximum lors d'une transition, il peut se propager 3 dépendances simultanées vers l'objet circuit. Toutes les cardinalités de propagation et donc de retrait de dépendance inférieures (0,1,2) sont également possibles. C'est pourquoi le système de transitions de l'attribut d'état opérationnel lié aux dépendances permet différentes incrémentations du compteur (`m:+1`,`mm:+2`, `mmm:+3`) et les décrémentations correspondantes (`d:-1`;`dd:-2`; `ddd:-3`). Le système de transition est montre en partie dans la figure 5.

Le déblocage opérationnel d'un objet lié à une ressource réelle ne peut se faire qu'au travers de la ressource. Nous avons donc modélisé en plus un système de transitions simple permettant de faire basculer une ressource dans un état inopérant et de la sortir de cet état. Ce système de transitions est le système `Ressource` défini dans la figure 6. Ce système de transitions a été mis en place pour les objets `equipment` et `trailTerminationPointBidirectional`.

```
transition_system Ressource;

ok  |-e   -> ok,
      off -> out;
out |-e   -> out,
      on  -> ok;

<initial = {ok}>.
```

FIG. 6 – *Le système de transitions étiquetées de ressources physiques*

3.2 Le premier modèle

Le premier modèle que nous avons réalisé spécifie la configuration présentée dans la figure 7. Elle concerne les objets constituant les circuits (`interExchangeCircuit` et `circuitSubGroup`) et leurs points de terminaison (`circuitXtp` et `circuitSubGroupTerminationPoint`) et met en œuvre les dépendances de connectivité et de groupe. Cette configuration est indépendante dans le sens où toute action de gestion appliquée à l'un de ses objets n'a de répercution que dans des objets de la même configuration. Il est certain que des opérations sur d'autres objets du modèle pourraient avoir une conséquence sur ces objets, nous étendrons partiellement cette configuration dans le second modèle. Le but ici est de construire un système de transitions réunissant ces différents objets.

La structure du système de synchronisation associé est présenté dans la figure 8. Comme la configuration comporte 6 objets à deux attributs chacun on retrouve un système de synchronisation à 12 systèmes de

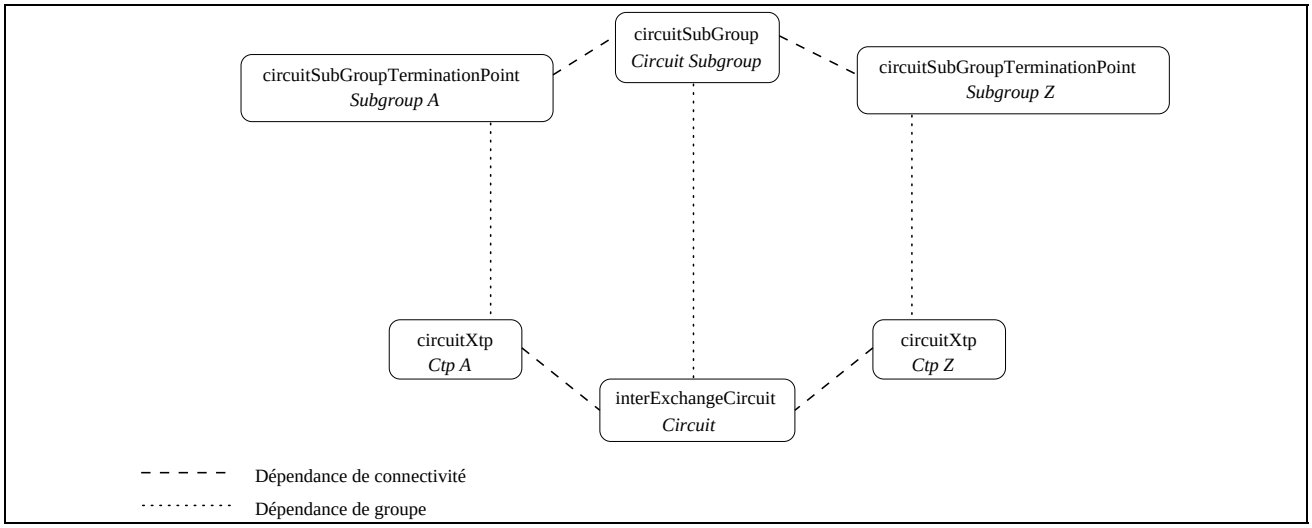

FIG. 7 – *Configuration de circuits*

```
synchronization_system term
<width=12;
list=(AdministrativeState, OperationalStateDependency,        \* Subgroup A        *\
      AdministrativeState, OperationalStateDependency,        \* Ctp A             *\
      AdministrativeState, OperationalStateDependency,        \* Circuit Subgroup  *\
      AdministrativeState, OperationalStateDependency,        \* Circuit           *\
      AdministrativeState, OperationalStateDependency,        \* Ctp Z             *\
      AdministrativeState, OperationalStateDependency)>;      \* Subgroup Z        *\
...
```

FIG. 8 – *Le système de synchronisation issu du premier modèle*

transitions, chaque couple (`AdministrativeState`, `OperationalStateDependency`) représentant un objet du modèle. L'attribut d'état de la ressource sous-jacente (`ressource`) n'est présent dans aucun objet, ceux-ci ne représentant que des objets logiques.

Sur ce système de synchronisation, nous avons spécifié l'ensemble des vecteurs de synchronisation qui modélisent les dépendances et transitions possibles dans le modèle. Le nombre de vecteurs étant élevé (24), nous n'en détaillons que 2 dans la section. Ces deux exemples sont donnés dans la figure 9.

Le premier vecteur modélise l'opération de blocage administratif (`lock` ligne (1)) sur un point de terminaison (ici le point A). Cette opération a pour conséquence d'ajouter une dépendance sur le circuit attaché (`disablepDep` au `circuit` ligne (4)) ainsi que sur le point de terminaison opposé (`disablepDep` au `Ctp Z` ligne (5)). Cette opération n'influence aucun autre objet du modèle.

Le second vecteur modélise l'opération de blocage d'un sous-groupe de circuits (action `lock` sur un `Circuit Subgroup` ligne (3)). Cette action a des effets directs et indirects sur d'autres objets. Les effets **directs** sont:

- blocage administratif et opérationnel des points de terminaison du sous-groupe de circuits (actions `lock` et + 1 `Dep` sur `SubgroupA` et `SubgroupZ` lignes (1) et (6)).
- ajout d'une dépendance au circuit sous-jacent (+1 `Dep` sur l'objet `circuit` ligne (4)).

Les actions **indirectes** sont dûes au blocage administratif des points de terminaison de sous-groupes de circuits. En effet, ce blocage entraîne une propagation de dépendances vers les points de terminaison de circuit sous-jacents (+1 `Dep` sur chaque `circuitXtp` lignes (2) et (5)). Cet ajout de dépendances va également se propager le long de la relation de connectivité et ce pour chaque point de terminaison. Cela implique que le point de terminaison A va propager une dépendance vers le circuit ainsi que vers le point de terminaison Z. Le point de terminaison Z va en parallèle déclencher la même propagation vers le point A et le circuit. Ce qui explique la présence de deux dépendances (`disableppDep`) dans les points de terminaison A et Z aux lignes (2) et (5) et de trois (`disablepppDep`) dans le circuit à la ligne (4).

```
       \* lock Ctp A: Rule 1 *\
(1)     (e.    e                \* subgroup A       *\
(2)     .lock. e                \* Ctp A            *\
(3)     .e.    e                \* Circuit Subgroup *\
(4)     .e.    disablepDep      \* Circuit          *\
(5)     .e.    disablepDep      \* Ctp Z            *\
(6)     .e.    e                \* subgroup Z       *\
       );
...
        \* lock CircuitSubgroup on unlocked Subgroup TPs: Rule 2 & 4 *\
(1)     (lock. disableppDep     \* subgroup A       *\
(2)     .e.    disableppDep     \* Ctp A            *\
(3)     .lock. disableppDep     \* Circuit Subgroup *\
(4)     .e.    disablepppDep    \* Circuit          *\
(5)     .e.    disableppDep     \* Ctp Z            *\
(6)     .lock. disableppDep     \* subgroup Z       *\
        );
...
```

FIG. 9 – *Quelques vecteurs de synchronisation du premier modèle*

De plus, le blocage administratif des points de terminaison de sous groupe de circuits entraîne une propagation des dépendances sur la relation de connectivité entre ces points et le sous-groupe de circuits qui les lie. De ce fait chaque point de terminaison de sous-groupe de circuit reçoit une dépendance de l'autre (`disableppDep` aux lignes (1) et (6)) et le sous-groupe de circuits en reçoit une de chaque point de terminaison de groupe de circuits, soit deux au total (`disableppDep` ligne (3)).

Le système de synchronisation comporte 24 vecteurs. Ces vecteurs spécifient l'ensemble des actions externes (blocage administratif, déblocage administratif) et leurs implications sur le système. Le produit synchronisé résultant comporte 64 états et 544 transitions. les vérifications entreprises sur ce modèle sont les mêmes que celles appliquées au modèle suivant. Elle sont décrites dans la section 3.4.

3.3 Le second modèle

La deuxième configuration permet de représenter cette fois toutes les relations citées dans la partie 2.2. Comme le montre la figure 10, chaque point de terminaison est contenu par un point de terminaison physique (`trailTerminationPointBidirectional`), lui même en relation de groupe avec un équipement (`equipment`).

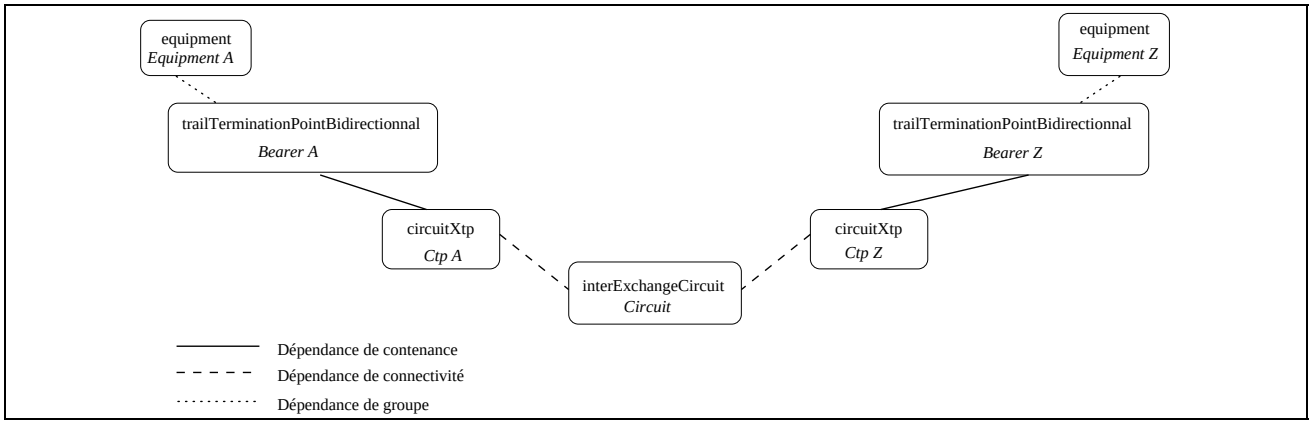

FIG. 10 – *Configuration exhaustive des relations*

Le système de synchronisation est donné dans la figure 11. Notons que les équipements et les points de terminaison physiques comportent trois systèmes de transitions alors que les points de terminaison de circuit et le ciruit n'en comportent que deux. Ceci est dû aux dépendances liées aux ressources réelles sous-jacentes aux équipements et aux points de terminaison physiques. C'est pourquoi nous avons rattaché à chacun d'eux un système de transitions de type `ressource` qui indique lorsqu'une ressource réelle devient inopérante et

```
synchronization_system term
<width=18;
list=(
AdministrativeState, OperationalStateDependency, Ressource,   \* Equipment A  *\
AdministrativeState, OperationalStateDependency, Ressource,   \* Bearer A     *\
AdministrativeState, OperationalStateDependency,              \* Ctp A        *\
AdministrativeState, OperationalStateDependency,              \* Circuit      *\
AdministrativeState, OperationalStateDependency,              \* Ctp Z        *\
AdministrativeState, OperationalStateDependency, Ressource,   \* Bearer Z     *\
AdministrativeState, OperationalStateDependency, Ressource)>; \* Equipment Z  *\
```

FIG. 11 – *Le système de sychronisation du second modèle*

inversement. On se retrouve donc sur ce modèle avec un système de synchronisation comportant 18 systèmes de transitions à synchroniser.

Tout comme pour le modèle précédent, nous détaillons dans cette section, deux vecteurs de transitions illustrés dans la figure 12.

```
       \* crash Equipment A: Rules: 4  *\
(1)    (e. e.           off      \* Equipment A *\
(2)    .e. disablepDep. e        \* Bearer A    *\
(3)    .e. disablepDep           \* Ctp A       *\
(4)    .e. disablepDep           \* Circuit     *\
(5)    .e. disablepDep           \* Ctp Z       *\
(6)    .e. e.           e        \* Bearer Z    *\
(7)    .e. e.           e        \* Equipment Z *\
       );
...
       \* unlock Circuit with Ctp A locked and Ctp Z unlocked: Rule 1 *\
(1)    (e.      e.          e    \* Equipment A *\
(2)    .e.      e.          e    \* Bearer A    *\
(3)    .unlock. enablemDep       \* Ctp A       *\
(4)    .unlock. enablemDep       \* Circuit     *\
(5)    .e.      enablemmDep      \* Ctp Z       *\
(6)    .e.e.               e    \* Bearer Z    *\
(7)    .e.e.               e    \* Equipment Z *\
       );
```

FIG. 12 – *Quelques vecteurs de synchronisation du second modèle*

Le premier vecteur de transitions illustre les conséquences d'une panne sur un équipement (passage à **off** ligne (1)) de l'état de la ressource. Celui-ci implique l'ajout d'une dépendance (**disablepDep**) sur l'ensemble de la chaîne (point de terminaison physique, point de terminaison de circuit, circuit, point de terminaison de circuit opposé, lignes (2)(3)(4) et (5)). Le passage à **on** de la ressource enlèvera l'ensemble de ces dépendances.

Le second vecteur spécifie les dépendances liées au déblocage d'un circuit lorsque l'un des points (**Ctp Z**) est déjà débloqué alors que l'autre (**Ctp A**) est encore bloqué . Dans tous les cas, le circuit est débloqué (action **unlock** ligne (4)) et une dépendance est enlevée sur chaque point de terminaison de circuit (**enablemDep** ligne (3)). De plus, le point de terminaison de circuit A est automatiquement débloqué (**unlock** ligne (3)). Cela entraîne un retrait d'une dépendance dans le circuit (**enablemDep** ligne (4)) et dans le point de terminaison opposé (ici Z, donc au total deux retrait de dépendance soit **enablemmDep** ligne (5)). En résumé, le déblocage du circuit entraîne le déblocage des points de terminaison qui lui même entraîne le retrait de dépendances.

Nous avons défini 28 vecteurs de synchronisation pour ce modèle. Le résultat comporte 2048 états et 25088 transitions. Les vérifications entreprises sur le modèle sont décrites dans la section suivante.

3.4 Les validations entreprises

Sur les deux modèles, nous avons entrepris les vérifications suivantes:

– absence de **deadlock** *(a)*

- possibilité de toujours revenir à l'état initial du système *(b)*
- absence d'états incohérents *(c)*
- limite maximale des dépendances *(d)*

Les formules **MEC** associées aux deux premières de ces vérifications sont définies ci-dessous.

```
deadlocks:=*-src(*)  (a)
```

```
retourStable:=*-coreach(initial,*)  (b)
```

Ces deux propriétés ont été vérifiées et le modèle ne présente pas d'erreurs (absence de deadlock et retour possible à un état opérationnel dans tous les cas).

En ce qui concerne l'absence d'états incohérents du système, il nous a, dans un premier temps, fallu identifier les caractéristiques de base d'un état incohérent. Dans le cadre d'un fonctionnement normal, nous en avons identifié un certain nombre et avons défini les formules **MEC** correspondantes. Ces états modélisent notamment des ressources de communication (points de terminaison, circuits) en fonctionnement reliés à des ressources bloquées ou inopérantes. Les formules associées sont donc des identifications d'états sur le produit synchronisé. Nous en donnons deux à titre d'exemple ci-dessous [4] :

- circuit opérationnel et non bloqué lié à au moins un point de terminaison bloqué ou inopérant:

  Sur le premier modèle:

```
pb1:= (((!state[6]='d*' \/ !state[10]='d*')
        \/ ((!state[5]='l*' \/ !state[9]='l*'))
        /\ (!state[7]='u*' /\ !state[8]='e*'));
```

  Sur le second modèle:

```
pb1:= (((!state[8]='d*' \/ !state[12]='d*')
        \/ ((!state[7]='l*' \/ !state[11]='l*'))
        /\ (!state[9]='u*' /\ !state[10]='e*'));
```

- circuit inopérant lié à deux points de terminaison opérationnels et non bloqués:

  Sur le premier modèle:

```
pb2:= (((!state[6]='e*' /\ !state[10]='e*')
        /\ (!state[5]='u*' /\ !state[9]='u*'))
        /\ (!state[7]='l*' \/ !state[8]='d*'));
```

  Sur le second modèle:

```
pb2:= (((!state[8]='e*' /\ !state[12]='e*')
        /\ (!state[7]='u*' /\ !state[11]='u*'))
        /\ (!state[9]='l*' \/ !state[10]='d*'));
```

D'autres vérifications ont été entreprises sur ce principe (ex. circuit opérationnel lié à des ressources bloquées) et toutes les dépendances ont été vérifiées. De même, nous avons vérifié la cohérence de notre contrainte de dépendance maximale en ajoutant un dégré supplémentaire à la dépendance et en vérifiant que l'état associé n'était jamais atteint.

4. !state[n]='d*' signifiant: *le $n^{ième}$ système de transition est dans un état dont la valeur commence par un d*

4 Résultats obtenus

Sur la base du modèle défini dans le document du NM/Forum, nous avons établi plusieurs schémas de systèmes de transitions. Le premier, non présenté ici, était basé sur une interprétation stricte des règles de dépendance spécifiées dans la norme. Le second était construit sur la base d'une rédéfinition des règles et avait pour but de vérifier la cohérence de notre nouveau modèle.

Le premier modèle a permis d'identifier des incohérences dans les règles de la norme et de les étendre. Ces incohérences concernent, d'une part la propagation des dépendances mais également l'oubli d'un lien supplémentaire entre le point de terminaison physique et le point de terminaison de circuit dans les règles de la norme ce qui avait pour effet de ne pas propager des dépendances cruciales.

L'extension des règles a été nécessaire pour avoir une description complète des dépendances entre les systèmes, ce qui n'était pas le cas dans le modèle initial. Sur la base de cette extension, nous avons redéfini un modèle que nous avons validé en deux étapes à l'aide de `MEC`. A partir de ce modèle, nous avons spécifié à nouveau les règles comportementales en y incluant les propagations systématiques. En raison de la taille du modèle nous avons dû le diviser en deux sous-systèmes et les valider séparément en raison des besoins en ressources machine trop élévées pour `MEC` dans notre modèle complet.

Nous avons également spécifié le modèle sur la base des scénarios fournis dans la norme. Ce modèle n'est pas présenté dans cet article. Neanmoins, la validation des scénarios y a fait apparaître des erreurs (transitions absentes). En effet la prise en compte des transitions telles que spécifiées dans les exemples pouvait mener à des situations de `''deadlock''`.

Les résultats de ce travail nous révèlent deux points importants. Le premier concerne la trop grande liberté d'interprétation d'une spécification informelle d'un modèle de l'information. En effet, malgré les efforts entrepris par les concepteurs du catalogue qui est très détaillé, il y reste des erreurs ou des oublis ainsi que des incohérences des règles par rapport aux scénarios donnés dans le document. Ce sont ces oublis qui permettent d'interpréter le comportement d'au moins 3 manières différentes, chacune de ces interprétations menant, à cause des erreurs, à des modèles incohérents. Pour preuve de cette liberté d'interprétation, les auteurs ont initialement construit deux modèles complètement différents à partir des mêmes règles.

Le second point important concerne l'absence d'information sur les valeurs initiales d'une configuration et sur les conséquences d'une instanciation ou d'une destruction d'un des objets du modèle. Nous avons pour cela, défini plusieurs spécifications avec des états initiaux différents. On arrive ici à la limite de l'utilisation de `MEC` car l'on a de nombreux aspects dynamiques liées à la création et à la destruction d'objets (ex. destruction d'un circuit et implication sur les points de terminaison de circuit, ...). Ce type de modélisation est beaucoup plus complexe en `MEC` qu'en LDS par exemple.

Les principaux problèmes identifiés lors de cette étude ont été, en plus de la nécessité de construire sur un niveau l'ensemble des propagations comportementales de pouvoir décomposer le système en un ensemble de systèmes plus simples et de définir les propriétés attendues comme cela avait déjà été mentionné dans [Arnold 93]. Une fois définies, l'expression des propriétés en `MEC` n'a pas posé de problème. Une approche différente aurait été de modéliser notre systèmes avec des états intermédiaires pour décrire de manière explicite des délais dans la propagation des dépendances.

5 Conclusion et travaux futurs

Dans cet article nous avons présenté une modélisation d'un ensemble d'objets de gestion inter-dépendants sous forme de systèmes de transitions étiquetées ainsi que la validation du modèle à l'aide du logiciel `MEC`. D'une part, cette étude nous a permis d'identifier des erreurs et oublis dans la norme et de les rectifier. Ceci nous encourage à poursuivre nos travaux dans le domaine en nous confirmant le besoin de décrire de manière formelle les comportements. D'autre part, ce travail a montré que le logiciel `MEC` se montrait utile pour la vérification des propriétés que nous avions définies pour notre modèle. Cependant, la nécessité de "mettre à plat" l'ensemble des propagations afin de les décrire dans un seul vecteur de synchronisation peut très vite devenir très complexe et qu'un langage de plus haut niveau serait peut-être mieux adapté. Il faut ajouter ici que les objets considérés comportent bien plus d'attributs que ceux retenus dans notre modélisation. Cependant, une vérification ne considérant que les attributs impliqués dans des dépendances est possible et suffisante pour valider le modèle.

Ce travail s'inscrit dans une activité plus large au sein du groupe RESEDAS, activité de recherche visant à fournir aux concepteurs de bases d'informations de gestion les moyens de décrire de manière formelle, précise et complète, leurs modèles. Nous nous sommes consacrés dans un premier temps au comportement au sein des

objets. Nous travaillons actuellement sur la spécification d'un ensemble d'objets et sur leurs dépendances ceci à différents niveaux d'abstraction.

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.

[Arnold 93] A. Arnold et S. Brlek. *"Conception d'un système de gestion d'appels téléphoniques: un exemple d'utilisation de me'thodes formelles"*. 1993. CFIP'93.

[CCITT.M.3010 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Principles for a Telecommunications Management Network"*, International Standard, CCITT.M.3010, Janvier 1992.

[CCITT.M.3100 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Generic Network Information Model"*, Draft International Standard, CCITT.M.3100, Janvier 1992.

[ISO-10164.3 92] International Organization for Standardization (ISO), *"System Management - Part 3: Attributes for Representing Relationships"*, International Standard, ISO-10164.3, 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.

[ISO-7498.4 89] International Organization for Standardization (ISO), *"Basic Reference Model - Part 4: Management framework"*, International Standard, ISO-7498.4, November 1989.

[ISO-8824 90] International Organization for Standardization (ISO), *"Specification of Abstract Syntax Notation Number One (ASN.1)"*, International Standard, ISO-8824, December 1990.

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

6 Annexes

6.1 Une fenêtre `MEC`

La figure 13 comporte un exemple de l'utilisation du logiciel **`MEC`**.

Elle comporte le résultat de la synchronisation du premier modèle ainsi que les valeurs de quelques variables calculées.

6.2 La spécification du compteur de dépendances

Ci-dessous le lecteur trouvera la spécification en **`MEC`** de l'attribut d'état opérationnel.

```
transition_system OperationalStateDependency;

enabled      |-disablepDep   -> disabledDep,
               disableppDep  -> disabledDep2,
               disablepppDep -> disabledDep3
               e             -> enabled;

disabledDep  |-enablemDep    -> enabled,
               disablepDep   -> disabledDep2,
               disableppDep  -> disabledDep3,
               disablepppDep -> disabledDep4,
               e             -> disabledDep;
```

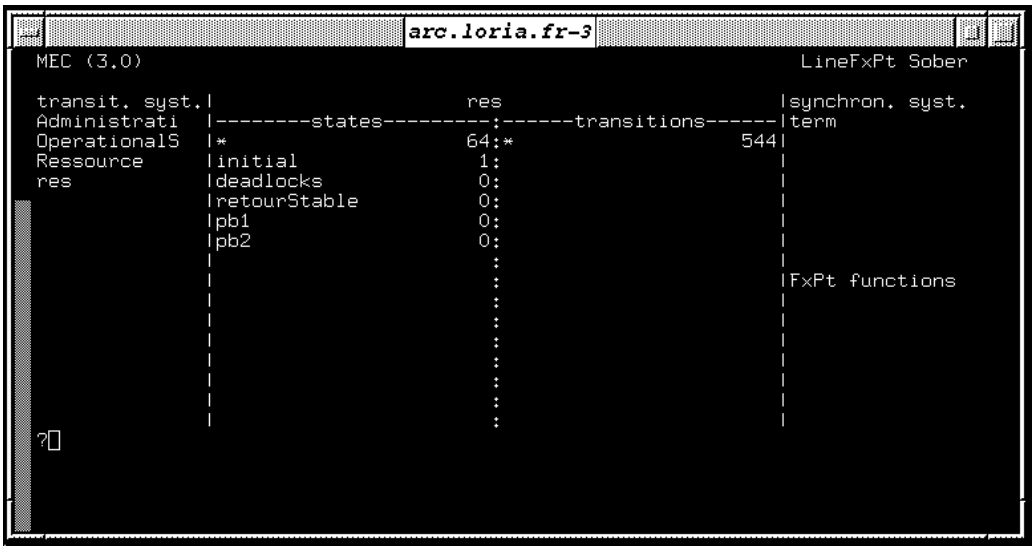

FIG. 13 – *Un exemple d'utilisation du logiciel* **MEC**

```
disabledDep2 |-enablemDep    -> disabledDep,
               enablemmDep   -> enabled,
               disablepDep   -> disabledDep3,
               disableppDep  -> disabledDep4,
               disablepppDep -> disabledDep5,
               e             -> disabledDep2;

disabledDep3 |-enablemDep    -> disabledDep2,
               enablemmDep   -> disabledDep,
               enablemmmDep  -> enable,
               disablepDep   -> disabledDep4,
               disableppDep  -> disabledDep5,
               disablepppDep -> disabledDep6,
               e             -> disabledDep3;

disabledDep4 |-enablemDep    -> disabledDep3,
               enablemmDep   -> disabledDep2,
               enablemmmDep  -> disabledDep,
               disablepDep   -> disabledDep5,
               disableppDep  -> disabledDep6,
               disablepppDep -> disabledDep7,
               e             -> disabledDep4;

disabledDep5 |-enablemDep    -> disabledDep4,
               enablemmmDep  -> disabledDep2,
               enablemmDep   -> disabledDep3,
               disablepDep   -> disabledDep6,
               disableppDep  -> disabledDep7,
               disablepppDep -> disabledDep8,
               e             -> disabledDep5;

disabledDep6 |-enablemDep    -> disabledDep5,
               enablemmDep   -> disabledDep4,
               enablemmmDep  -> disabledDep3,
               disableppDep  -> disabledDep8,
```

```
               disablepppDep -> disabledDep9,
               disablepDep   -> disabledDep7,
               e             -> disabledDep6;

disabledDep7 |-enablemDep    -> disabledDep6,
               enablemmDep   -> disabledDep5,
               enablemmmDep  -> disabledDep4,
               disableppDep  -> disabledDep9,
               disablepppDep -> disabledD10,
               disablepDep   -> disabledDep8,
               e             -> disabledDep7,

disabledDep8 |-enablemDep    -> disabledDep7,
               enablemmDep   -> disabledDep6,
               enablemmmDep  -> disabledDep5,
               disableppDep  -> disabledD10,
               disablepDep   -> disabledDep9,
               e             -> disabledDep8;

disabledDep9 |-enablemDep    -> disabledDep8,
               enablemmDep   -> disabledDep7,
               enablemmmDep  -> disabledDep6,
               disablepDep   -> disabledD10,
               e             -> disabledDep9;

disabledD10   |-enablemDep   -> disabledDep9,
                enablemmDep  -> disabledDep8,
                enablemmmDep -> disabledDep7,
                e            -> disabledD10;

<initial = enabled>.
```

6.3 Le premier système

Ci-dessous le lecteur trouvera la spécification du premier modèle. Le système de synchronisation comporte 14 systèmes de transitions et 24 vecteurs de synchonisation.

```
synchronization_system term
<width=12;
list=(AdministrativeState, OperationalStateDependency,        \* Subgroup A        *\
      AdministrativeState, OperationalStateDependency,        \* Ctp A             *\
      AdministrativeState, OperationalStateDependency,        \* Circuit Subgroup  *\
      AdministrativeState, OperationalStateDependency,        \* Circuit           *\
      AdministrativeState, OperationalStateDependency,        \* Ctp Z             *\
      AdministrativeState, OperationalStateDependency)>;      \* Subgroup Z        *\

\* lock Ctp A: Rule 1 *\
(e.      e             \* subgroup A       *\
.lock.   e             \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.e.      disablepDep   \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock Ctp Z: Rule 1 *\
(e.      e             \* subgroup A       *\
```

```
.e.      disablepDep   \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.e.      disablepDep   \* Circuit          *\
.lock.   e             \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Ctp A: Rule 1 (complementary)    *\
(e.      e             \* subgroup A       *\
.unlock. e             \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.e.      enablemDep    \* Circuit          *\
.e.      enablemDep    \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Ctp Z: Rule 1 (complementary)    *\
(e.      e             \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.e.      enablemDep    \* Circuit          *\
.unlock. e             \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock Circuit on unlocked Ctps: Rule 1   *\
(e.      e             \* subgroup A       *\
.lock.   disableppDep  \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.lock.   disableppDep  \* Circuit          *\
.lock.   disableppDep  \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Circuit on locked Ctps: Rule 1   *\
(e.      e             \* subgroup A       *\
.unlock. enablemmDep   \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.unlock. enablemmDep   \* Circuit          *\
.unlock. enablemmDep   \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock Circuit on locked CtpZ: Rule 1 *\
(e.      e             \* subgroup A       *\
.e.      disablepDep   \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.lock.   e             \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Circuit on unlocked CtpZ: Rule 1 *\
(e.      e             \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.unlock. e             \* Circuit          *\
```

```
.e.      enablemDep    \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock Circuit with Ctp A locked and Ctp Z unlocked: Rule 1 *\
(e.      e             \* subgroup A       *\
.e.      disableppDep  \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.lock.   disablepDep   \* Circuit          *\
.lock.   disablepDep   \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock Circuit with Ctp Z locked and Ctp A unlocked: Rule 1 *\
(e.      e             \* subgroup A       *\
.lock.   disablepDep   \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.lock.   disablepDep   \* Circuit          *\
.e.      disableppDep  \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Circuit with Ctp A locked and Ctp Z unlocked: Rule 1 *\
(e.      e             \* subgroup A       *\
.unlock. enablemDep    \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.unlock. enablemDep    \* Circuit          *\
.e.      enablemmDep   \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* unlock Circuit with Ctp Z locked and Ctp A unlocked: Rule 1 *\
(e.      e             \* subgroup A       *\
.e.      enablemmDep   \* Ctp A            *\
.e.      e             \* Circuit Subgroup *\
.unlock. enablemDep    \* Circuit          *\
.unlock. enablemDep    \* Ctp Z            *\
.e.      e             \* subgroup Z       *\
);

\* lock CircuitSubgroup on unlocked Subgroup TPs: Rule 2 & 4 *\
(lock.   disableppDep  \* subgroup A       *\
.e.      disableppDep  \* Ctp A            *\
.lock.   disableppDep  \* Circuit Subgroup *\
.e.      disablepppDep \* Circuit          *\
.e.      disableppDep  \* Ctp Z            *\
.lock.   disableppDep  \* subgroup Z       *\
);

\* unlock CircuitSubgroup on locked Circuit Subgroups: Rule 2 & 4 *\
(unlock. enablemmDep   \* subgroup A       *\
.e.      enablemmDep   \* Ctp A            *\
.unlock. enablemmDep   \* Circuit Subgroup *\
.e.      enablemmmDep  \* Circuit          *\
.e.      enablemmDep   \* Ctp Z            *\
.unlock. enablemmDep   \* subgroup Z       *\
);
```

```
\* lock Subgroup A: Rule 1 & Rule 4 & Rule 5 *\
(lock.   e             \* subgroup A       *\
.e.      disablepDep   \* Ctp A            *\
.e.      disablepDep   \* Circuit Subgroup *\
.e.      disablepDep   \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.e.      disablepDep   \* subgroup Z       *\
);

\* lock Subgroup Z: Rule 1 & Rule 4 & Rule 5 *\
(e.      disablepDep   \* subgroup A       *\
.e.      disablepDep   \* Ctp A            *\
.e.      disablepDep   \* Circuit Subgroup *\
.e.      disablepDep   \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.lock.   e             \* subgroup Z       *\
);

\* unlock Subgroup A: Rule 1 (complementary) & Rule 4 & Rule 5 *\
(unlock. e             \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.e.      enablemDep    \* Circuit Subgroup *\
.e.      enablemDep    \* Circuit          *\
.e.      enablemDep    \* Ctp Z            *\
.e.      enablemDep    \* subgroup Z       *\
);

\* unlock Subgroup Z: Rule 1 (complementary) & Rule 4 & Rule 5 *\
(e.      enablemDep    \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.e.      enablemDep    \* Circuit Subgroup *\
.e.      enablemDep    \* Circuit          *\
.e.      enablemDep    \* Ctp Z            *\
.unlock. e             \* subgroup Z       *\
);

\* lock CircuitSubGroup on locked Subgroup Tps:  Rule 4 *\
(e.      disablepDep   \* subgroup A       *\
.e.      e             \* Ctp A            *\
.lock.   e             \* Circuit Subgroup *\
.e.      disablepDep   \* Circuit          *\
.e.      e             \* Ctp Z            *\
.e.      disablepDep   \* subgroup Z       *\
);

\* unlock CircuitSubGroup on unlocked Subgroup Tps: Rule 4 *\
(e.      enablemDep    \* subgroup A       *\
.e.      e             \* Ctp A            *\
.unlock. e             \* Circuit Subgroup *\
.e.      enablemDep    \* Circuit          *\
.e.      e             \* Ctp Z            *\
.e.      enablemDep    \* subgroup Z       *\
);

\* lock CircuitSubGroup with A Subgroup TP locked and Z Subgroup TP unlocked:  Rule 4 *\
(e.      disableppDep  \* subgroup A       *\
```

```
.e.      disablepDep   \* Ctp A            *\
.lock.   disablepDep   \* Circuit Subgroup *\
.e.      disableppDep  \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.lock.   disablepDep   \* subgroup Z       *\
);

\* unlock CircuitSubGroup with A Subgroup TP locked and Z Subgroup TP unlocked:  Rule 4 *\
(unlock. enablemDep    \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.unlock. enablemDep    \* Circuit Subgroup *\
.e.      enablemmDep   \* Circuit          *\
.e.      enablemDep    \* Ctp Z            *\
.e.      enablemmDep   \* subgroup Z       *\
);

\* lock CircuitSubGroup with Z Subgroup TP locked and A Subgroup TP unlocked:  Rule 4 *\
(lock.   disablepDep   \* subgroup A       *\
.e.      disablepDep   \* Ctp A            *\
.lock.   disablepDep   \* Circuit Subgroup *\
.e.      disableppDep  \* Circuit          *\
.e.      disablepDep   \* Ctp Z            *\
.e.      disableppDep  \* subgroup Z       *\
);

\* unlock CircuitSubGroup with Z Subgroup TP locked and A Subgroup TP unlocked:  Rule 4 *\
(e.      enablemmDep   \* subgroup A       *\
.e.      enablemDep    \* Ctp A            *\
.unlock. enablemDep    \* Circuit Subgroup *\
.e.      enablemmDep   \* Circuit          *\
.e.      enablemDep    \* Ctp Z            *\
.unlock. enablemDep    \* subgroup Z       *\
).

sync(term,res);
dts(res);
```

6.4 Le second système

Ci-dessous le lecteur trouvera la spécification du second modèle. Le système de synchronisation comporte 18 systèmes de transitions et 28 vecteurs de synchonisation.

```
synchronization_system term
<width=18;
list=(AdministrativeState, OperationalStateDependency, Ressource,   \* Equipment A *\
      AdministrativeState, OperationalStateDependency, Ressource,   \* Bearer A   *\
      AdministrativeState, OperationalStateDependency,              \* Ctp A      *\
      AdministrativeState, OperationalStateDependency,              \* Circuit    *\
      AdministrativeState, OperationalStateDependency,              \* Ctp Z      *\
      AdministrativeState, OperationalStateDependency, Ressource,   \* Bearer Z   *\
      AdministrativeState, OperationalStateDependency, Ressource)>; \* Equipment Z *\

\* crash Equipment A: Rules: 4  *\
(e.      e.            off   \* Equipment A *\
.e.      disablepDep.  e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
```

```
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* enable Equipment A: Rules: 4  *\
(e.      e.            on    \* Equipment A *\
.e.      enablemDep.   e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* crash Equipment Z: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      disablepDep.  e     \* Bearer Z    *\
.e.      e.            off   \* Equipment Z *\
);

\* enable Equipment Z: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      enablemDep.   e     \* Bearer Z    *\
.e.      e.            on    \* Equipment Z *\
);

\* crash Bearer A: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
.e.      e.            off   \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* enable Bearer A: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
.e.      e.            on    \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* crash Bearer Z: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
```

```
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            off   \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* enable Bearer Z: Rules: 4  *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            on    \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Equipment A: Rules: 4  *\
(lock.   e.            e     \* Equipment A *\
.e.      disablepDep.  e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Equipment Z *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      disablepDep.  e     \* Bearer Z    *\
.lock.   e.            e     \* Equipment Z *\
);

\* unlock Equipment A *\
(unlock. e.            e     \* Equipment A *\
.e.      enablemDep.   e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Equipment Z *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      enablemDep.   e     \* Bearer Z    *\
.unlock. e.            e     \* Equipment Z *\
);
```

```
\* lock Bearer A: Rules:  *\
(e.      e.            e     \* Equipment A *\
.lock.   e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Bearer Z *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.lock.   e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Bearer A: Rules:  *\
(e.      e.            e     \* Equipment A *\
.unlock. e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* equipment Z *\
);

\* unlock Bearer Z *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.unlock. e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Ctp A: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.lock.   e                   \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Ctp Z: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.e.      disablepDep         \* Circuit     *\
.lock.   e                   \* Ctp Z       *\
```

```
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Ctp A: Rule 1 (complementary) *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.unlock. e                   \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Ctp Z: Rule 1 (complementary) *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemDep          \* Ctp A       *\
.e.      enablemDep          \* Circuit     *\
.unlock. e                   \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Circuit on unlocked Ctps: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.lock.   disableppDep        \* Ctp A       *\
.lock.   disableppDep        \* Circuit     *\
.lock.   disableppDep        \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Circuit on locked Ctps: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.unlock. enablemmDep         \* Ctp A       *\
.unlock. enablemmDep         \* Circuit     *\
.unlock. enablemmDep         \* Ctp Z       *\
.e.      e.            e     \* bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Circuit on locked CtpZ: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disablepDep         \* Ctp A       *\
.lock.   e                   \* Circuit     *\
.e.      disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Circuit on unlocked CtpZ: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
```

```
.e.      enablemDep          \* Ctp A       *\
.unlock. e                   \* Circuit     *\
.e.      enablemDep          \* Ctp Z       *\
.e.      e.            e     \* bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Circuit with Ctp A locked and Ctp Z unlocked: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      disableppDep        \* Ctp A       *\
.lock.   disablepDep         \* Circuit     *\
.lock.   disablepDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* lock Circuit with Ctp Z locked and Ctp A unlocked: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.lock.   disablepDep         \* Ctp A       *\
.lock.   disablepDep         \* Circuit     *\
.e.      disableppDep        \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Circuit with Ctp A locked and Ctp Z unlocked: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.unlock. enablemDep          \* Ctp A       *\
.unlock. enablemDep          \* Circuit     *\
.e.      enablemmDep         \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
);

\* unlock Circuit with Ctp Z locked and Ctp A unlocked: Rule 1 *\
(e.      e.            e     \* Equipment A *\
.e.      e.            e     \* Bearer A    *\
.e.      enablemmDep         \* Ctp A       *\
.unlock. enablemDep          \* Circuit     *\
.unlock. enablemDep          \* Ctp Z       *\
.e.      e.            e     \* Bearer Z    *\
.e.      e.            e     \* Equipment Z *\
).

sync(term,res);
dts(res);
```

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