Architectures de contrôle-commande redondantes à base d'Ethernet Industriel : modélisation et validation par model-checking temporisé

Steve Limal

N° ENSC-2009/142

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Monsieur Steve LIMAL

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine :
ELECTRONIQUE–ELECTROTECHNIQUE–AUTOMATIQUE

Sujet de la thèse :

Architectures de contrôle-commande redondantes à base d'Ethernet Industriel : Modélisation et validation par model-checking temporisé.

Thèse présentée et soutenue à Cachan le 8 janvier 2009 devant le jury composé de :

Francis LEPAGE	Professeur-Univ. H. Poincaré, Nancy-CRAN	Rapporteur
François VERNADAT	Professeur-INSA de Toulouse-LAAS	Rapporteur
Jean-Pierre ELLOY	Professeur-Ecole Centrale de Nantes-IRCCyN	Examinateur
Hervé SABOT	Responsable du service Engineering & Commissioning tool - Alstom Power	Examinateur invité
Jean-Jacques LESAGE	Professeur-ENS Cachan-LURPA	Directeur de thèse
Bruno DENIS	Maître de Conférence-ENS Cachan-LURPA	Co-encadrant

Laboratoire Universitaire de Recherche en Production Automatisée
(ENS CACHAN / EA 1385)
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

En souvenir de Nau.

Remerciements

Les travaux de recherche présentés dans ce mémoire ont été réalisés dans le cadre d'une convention CIFRE entre le Laboratoire Universitaire de Recherche en Production Automatisée (LURPA - EA 1385) et la société Alstom Power - Control Systems.

Je tiens tout d'abord à remercier le Professeur Jean-Jacques Lesage pour avoir assumé la direction de mes travaux. Je remercie également Monsieur Bruno Denis pour avoir assuré l'encadrement scientifique de mon travail.

Je remercie Messieurs les Professeurs Francis Lepage et François Vernadat qui ont accepté d'être rapporteurs de cette thèse. Je remercie également le Professeur Jean-Pierre Elloy et Monsieur Hervé Sabot pour avoir participé au jury.

Je remercie bien sûr la société Alstom Power pour avoir initié et encouragé ces travaux. Je remercie tout d'abord Messieurs Denis Cantero et Christian Kervenec pour m'avoir accordé leur confiance. Je remercie Messieurs Ludovic Guillemaud et Stéphane Potier pour avoir assuré l'encadrement industriel de mes travaux. Mes remerciements vont également à Madame Sandrine Couffin et toutes les personnes côtoyées au B8 pour m'avoir fait bénéficier autant de leurs qualités humaines que de leurs compétences techniques.

Merci à tous les membres du LURPA dont la diversité (~~informaticien chef~~ philosophe, coach charismatique, industrielle, nouveaux permanents, extranjeros, ~~Vincent~~ informaticien personnel...) fut enrichissante à de nombreux égards.

Je remercie enfin tous les membres de ma famille qui ont contribué bien avant le début des travaux à ces résultats.

Table des matières

Acronymes

AFDX	Avionics Full-Duplex Switched Ethernet
AMN	Active Managing Node. Rôle d'un RMN
AMNI	Active Managing Node Indication. Trame EPL-HA
ASIC	Application-Specific Integrated Circuit
ASnd	Asynchronous Send. Type de trame EPL
CN	Controlled Node. Type de nœud EPL
CSMA-CD	Carrier Sense Multiple Access - Collision Detection
CTL	Computation Tree Logic
EMS	Energy Management System
EPL	Ethernet PowerLink
EPL-HA	Ethernet PowerLink High Availability extension
EPSG	Ethernet PowerLink Standardization Group
ERP	Enterprise Ressource Planning
EtherNet-IP	EtherNet-Industrial Protocol
FCS	Frame Check Sequence
FPGA	Field-Programmable Gate Array
IDS	Isochronous Diffusion Sphere. 3.3
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFG	Inter Frame Gap
IP	Internet Protocol
IRT	Isochronous Real-Time
LS	Link Selector. Fonction utilisée par EPL-HA
MN	Managing Node. Type de nœud EPL
OSI	Open Systems Interconnection
PReq	Poll Request . Type de trame EPL
PRes	Poll Response. Type de trame EPL

RAMS	Reliability, Availability, Maintenabiltity and Safety
RMN	Redundant Managing Node. Type de nœud EPL
SCNM	EPL Slot Communication Network Management
SIL	Safety Integrity Level
SMN	Stand-by Managing Node. Rôle d'un RMN
SNMP	Simple Network Management Protocol
SoA	Start of Asynchronous. Type de trame EPL
SoC	Start of Cycle. Type de trame EPL
TCP	Transmission Control Protocol
TCTL	Timed Computation Tree Logic
TRCM	Temps Réel à Contraintes Mixtes
TRCR	Temps Réel à Contraintes Relatives
TRCS	Temps Réel à Contraintes Strictes
UDP	User Datagram Protocol

Introduction générale

En bureautique comme en industrie, les besoins des utilisateurs de réseaux informatiques augmentent régulièrement. Dans la bureautique, le paramètre de la qualité de service le plus connu est le débit maximum de données autorisé par une solution de communication, filaire ou sans fil. La solution filaire dominante est Ethernet qui a été continuellement développé depuis les années 80. Pourtant, son adoption dans les applications industrielles de communication entre équipements d'entrées-sorties et équipements de traitement des données est relativement récente.

Jusqu'alors, ces applications utilisaient essentiellement des solutions dites de bus de terrain (par exemple, Interbus ou Sercos). Celles-ci prennent en compte des paramètres de qualité de service spécifiques à l'industrie tels que la réactivité ou le déterminisme. Ce n'est que récemment qu'Ethernet est devenu financièrement et techniquement intéressant. Il est désormais considéré comme le futur remplaçant des bus de terrain.

Ce travail s'intéresse à la solution industrielle de contrôle-commande de centrales de production d'énergie ALSPA CONTROPLANT développée par Alstom Power Systems. Le composant CONTROBLOC de la solution ALSPA CONTROPLANT regroupe les équipements de terrains et briques logicielles supportant les fonctions d'acquisition des entrées et émission des sorties, de traitement et de communication au plus près des processus. Ce travail est centré sur la fonction de communication du CONTROBLOC qui propose pour cela différents réseaux de terrain (dont WorldFIP, Profibus et Modbus-TCP).

Le CONTROBLOC doit intégrer un nouveau réseau de terrain qui satisfasse aux nouvelles exigences des clients sur les paramètres de qualité de services. Parmi celles-ci, l'exigence de réactivité est particulièrement contraignante, d'autant qu'elle s'accompagne d'une exigence de disponibilité. Une coupure des communications doit non seulement être compensée, mais elle doit l'être de manière à ce que l'exigence de réactivité continue d'être respectée. Avec certaines applications manufacturières, le passage en position de repli (non productif) engendré par une défaillance a une conséquence financière qui incite à rechercher une meilleure disponibilité. Avec une application de production d'énergie, les conséquences possibles des problèmes de communication sur la sécurité des biens et des personnes imposent de considérer la disponibilité comme composante de la sûreté de fonctionnement.

Nos travaux débutent à partir du choix du nouveau protocole de réseau de terrain répondant à l'exigence de réactivité et spécifiant les fonctions de disponibilité attendues. Notre objectif est d'accompagner Alstom Power Systems pour la validation de la spécification du protocole par rapport à des exigences fonctionnelles décrivant la disponibilité recherchée. La validation de ces exigences est un élément qui s'avère nécessaire pour pouvoir prétendre certifier la solution globale à un niveau minimum de sûreté de fonctionnement. Dans ce contexte de sûreté de fonctionnement, le recours à la vérification

formelle s'est imposé pour attester qu'une architecture donnée répond bien à toutes les exigences fonctionnelles ainsi qu'aux exigences de sûreté. Cet aspect a constitué l'un des apports majeurs du laboratoire à Alstom Power.

Le premier chapitre développe l'application considérée et détaille le cahier des charges de la nouvelle solution réseau du CONTROBLOC vis-à-vis des trois paramètres de qualité de service que nous avons retenus, la réactivité, le déterminisme et la disponibilité. Ce cahier des charges est alors confronté aux solutions commerciales ou universitaires utilisant Ethernet comme support.

Des solutions commerciales utilisant le support Ethernet et répondant aux paramètres de qualité de service établis pour notre application existent. Ce travail ne considère que l'une d'entre elles, Ethernet PowerLink (EPL), dont l'extension Ethernet PowerLink High Availability (EPL-HA) ambitionne de développer les propriétés de disponibilité sans sacrifier les paramètres de qualité de service du protocole de base.

Le second chapitre est consacré à la description d'EPL, et des fonctions spécifiées par l'extension EPL-HA. Il présente ensuite les exigences fonctionnelles de disponibilité que nous souhaitons obtenir par le biais de cette nouvelle solution.

Notre objectif est de vérifier si l'ensemble des exigences fonctionnelles de disponibilité est rempli par une architecture vis-à-vis de la spécification. Pour cela, nous utilisons un modèle de l'architecture réseau utilisant la solution EPL-HA ainsi que d'un modèle des exigences fonctionnelles attendues. La modélisation doit nous permettre de conclure sur les exigences par l'utilisation de la technique de vérification formelle appelée model-checking. Les processus de production d'énergie étant aussi variés que les architectures réseau qui les desservent, nous proposons une modélisation paramétrable, prenant en compte la topologie de l'architecture.

Le troisième chapitre considère dans un premier temps la technique de vérification formelle et l'outil associé que nous avons retenu. La démarche de modélisation d'une architecture et la modélisation des exigences fonctionnelles sous la forme de propriétés à vérifier sont alors traitées.

Le dernier chapitre nous permet d'appliquer notre démarche de modélisation et la vérification des propriétés traduites à deux architectures représentatives de l'utilisation possible du réseau de terrain. La première étant en fait une famille d'architecture, ce chapitre nous permet également de confronter la technique de vérification formelle choisie à la conséquence attendue d'explosion combinatoire. Les résultats obtenus en termes de temps de calcul et de mémoire nécessaire permettent de donner un aperçu du domaine d'utilisation possible de la démarche proposée.

Chapitre 1

Contexte

La cellule d'automatisation et plus particulièrement son réseau de communication sont décrits dans ce premier chapitre.
Pour cela, les exigences associées à son rôle ainsi que le cahier des charges qui en résulte sont traités. Ce chapitre s'intéresse ensuite aux différentes solutions basées sur Ethernet susceptibles de répondre à ce cahier des charges.

Sommaire

1.1 Description du système de contrôle-commande

Le système de contrôle-commande que nous considérons dans ce document est utilisé dans le domaine de la production d'énergie. Les figures 1.1 et 1.2 illustrent deux types de centrale thermique.

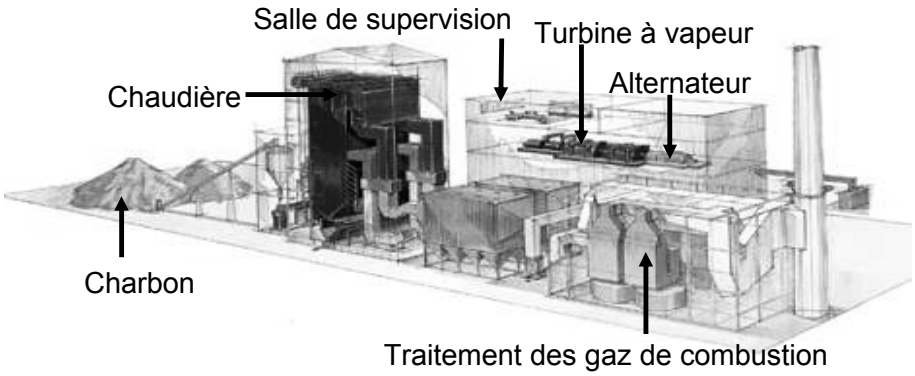


FIGURE 1.1 – Exemple de centrale thermique classique.

Les centrales thermiques classiques (figure 1.1) brûlent des énergies fossiles (gaz, fuel ou charbon) pour produire de la vapeur par l'intermédiaire d'une chaudière. La vapeur permet d'entraîner une turbine à vapeur reliée à un alternateur. Ce dernier transforme l'énergie mécanique d'entraînement en énergie électrique. Les centrales thermiques solaires ou à biomasse, géothermiques ou nucléaires utilisent d'autres sources de chaleur pour produire la vapeur.

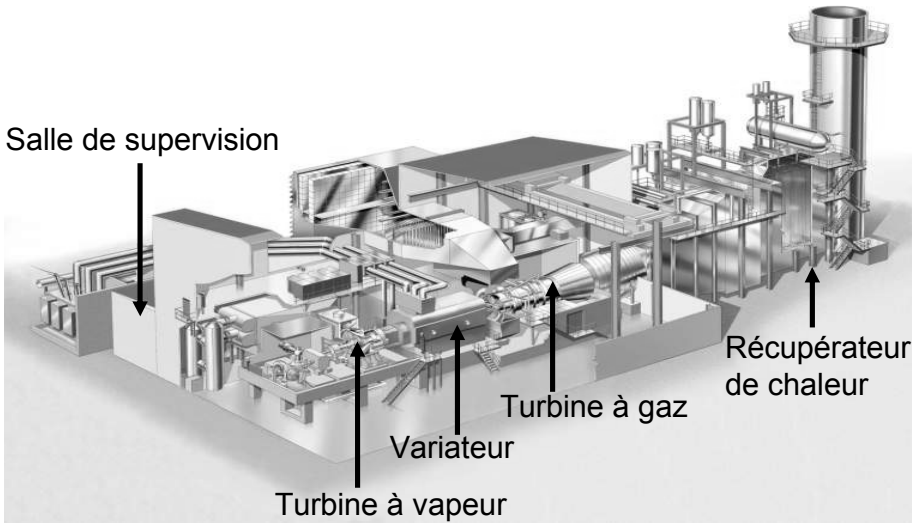


FIGURE 1.2 – Exemple de centrale thermique à cycle combiné.

Les centrales thermiques à cycle combiné (figure 1.2) utilisent un carburant (gaz ou fuel) pour entraîner une turbine à combustion reliée à un alternateur. Cet entraînement est combiné à celui fourni par une turbine à vapeur. Celle-ci est produite à partir des gaz d'échappement issus de la combustion. La combinaison permet d'améliorer le rendement

global de la production.

Les centrales hydroélectriques ou les centrales éoliennes utilisent la force générée par l'eau ou le vent pour entraîner une turbine. En dépit de leur variété, ces procédés sont tous automatisés. L'automatisation permet par exemple de piloter l'ouverture des vannes d'un barrage hydroélectrique comme de réguler l'alimentation en carburant d'une turbine à combustion. L'automatisation intervient également dans les processus associés comme la gestion des produits issus de la combustion ou le contrôle de la tension produite.

Par conséquent, une architecture de contrôle-commande est associée à un processus automatisé dans la centrale, que celui-ci soit réparti sur de grandes distances comme sur les barrages hydroélectriques ou les champs d'éoliennes, ou que les exigences de régulation soient relativement strictes comme en régulation. Sur des turbines à combustion de 330 tonnes, fournissant une puissance de 280 MWatt à un régime de 3000tr.min^{-1}, un défaut de régulation même court a des conséquences importantes sur la durée de vie de ce dispositif fortement sollicité.

Un exemple d'architecture du système de contrôle-commande ALSPA CONTROPLANT est donné dans la figure 1.3. D'après le standard IEEE 610.12 [73], une architecture est une organisation structurelle d'un système.

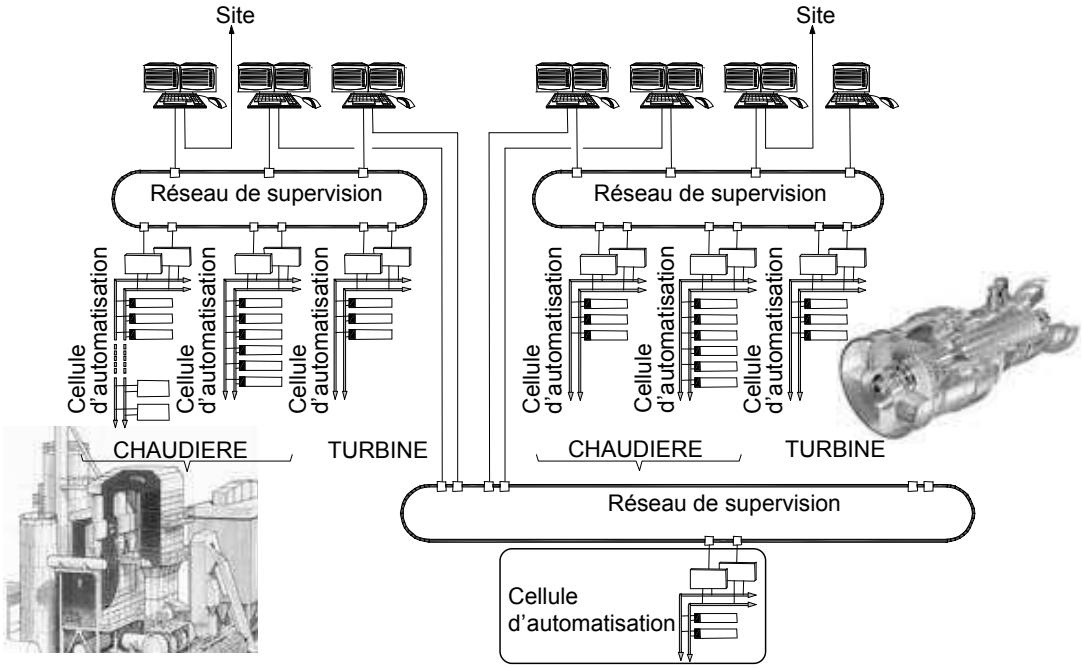


FIGURE 1.3 – Exemple du système de contrôle commande considéré (vue structurelle).

Dans ce document, nous employons le terme d'architecture pour désigner l'organisation structurelle du système contrôle-commande du point de vue du réseau de communication, c'est-à-dire les différents composants constituant le réseau de communication et leurs connexions physiques respectives. Les composants sont les nœuds réalisant les fonctions de contrôle-commande (acquisition et traduction des signaux relevés sur le processus, traitement et émission des commandes) mais aussi les composants d'interconnexion du réseau (câbles, répéteurs). Sur la figure 1.3, on peut remarquer que l'architecture utilise le même motif (réseau de supervision ou réseau de terrain) à différents emplacements

pour piloter différents processus liés à la chaudière et la turbine. En effet, le système de contrôle-commande a été défini pour offrir une solution unifiée, c'est-à-dire pour réaliser les systèmes d'automatisme ou le contrôle de machine tournante d'une centrale de production d'énergie avec les mêmes technologies.

Cette approche permet à une architecture du système de contrôle-commande donnée d'être décrite par la combinaison de postes d'ingénierie et de supervision, de réseaux de supervision ainsi que de cellules d'automatisation.

Le réseau de supervision relie des postes de supervision (situés principalement en salle de commande), des postes d'ingénierie (pour la programmation) ainsi que des cellules d'automatisation. On recense généralement un réseau de supervision par unité de production indépendante et plusieurs unités de production indépendantes par centrale. Pour des besoins de gestion (Enterprise Ressource Planning (ERP), Energy Management System (EMS)), les réseaux de supervision d'unités de production indépendantes peuvent être reliés au même réseau de site.

Les réseaux de supervision du système de contrôle-commande que nous considérons utilisent actuellement Ethernet commuté et une solution d'anneau. Ces technologies seront précisées respectivement dans les sections 1.4 et 1.5.

La cellule d'automatisation CONTROBLOC dont un exemple d'architecture est illustré sur la figure 1.4, est l'entité où sont automatisées une ou plusieurs parties du processus de production d'énergie. La cellule d'automatisation est un système de contrôle-commande distribué. Elle est constituée de nœuds (un contrôleur de cellule, des contrôleurs de terrain ou des équipements d'entrées-sorties déportées) communiquant par l'intermédiaire d'un réseau de communications appelé réseau de terrain. C'est aux caractéristiques et aux performances de ce réseau de terrain que nous nous intéressons exclusivement dans ce document.

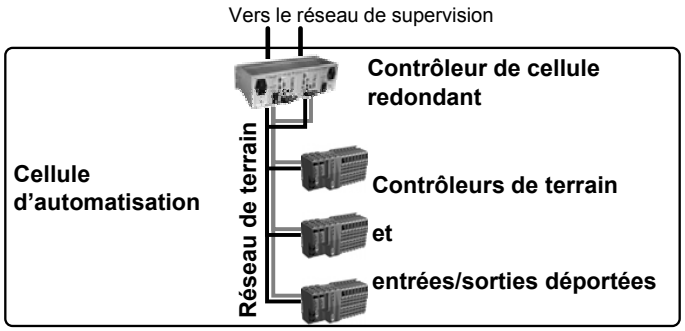


FIGURE 1.4 – Cellule d'automatisation (vue structurelle).

L'automatisation du processus de production d'énergie étant réalisée au niveau de la cellule, les caractéristiques à respecter pour maîtriser le pilotage du processus y sont localisées elles aussi et auront une composante sur le réseau de terrain. Les trois sous-sections suivantes détaillent les trois caractéristiques du réseau de terrain auxquelles nous nous intéressons spécialement dans ce document, la réactivité, le déterminisme ainsi que la disponibilité.

1.1.1 Exigence de réactivité.

Comme l'illustre la figure 1.5, il existe un délai de réaction entre le changement d'état d'un signal d'entrée acquis via un capteur sur un processus et sa conséquence sur un signal de sortie émis pour commander un préactionneur du processus. Cette illustration est inspirée de celle utilisée par Gaëlle Marsal dans son mémoire de thèse [1].

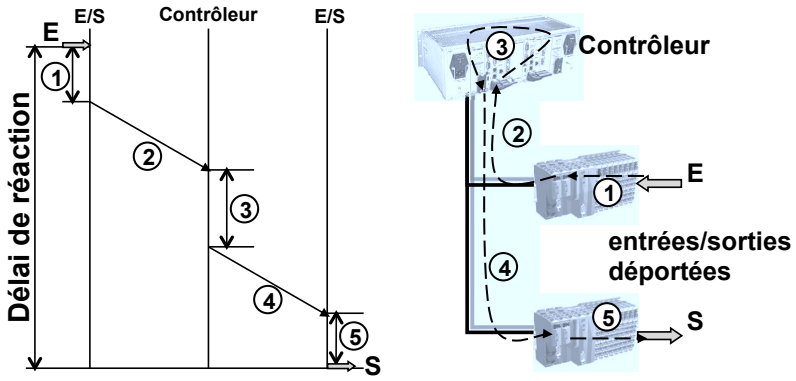


FIGURE 1.5 – Exemple illustrant le terme de délai de réaction

Dans cet exemple, le délai de réaction regroupe :

① le délai d'acquisition d'un signal d'entrée E sur le fond de panier d'un équipement d'entrées-sorties déportées ou un contrôleur de terrain,

② le délai de transmission à un contrôleur de cellule pour traitement,

③ le délai de traitement,

④ le délai de transmission à un équipement d'entrées-sorties (ou un contrôleur de terrain) tiers et

⑤ le délai de mise à jour d'un signal de sortie S par l'intermédiaire de son fond de panier.

La cellule d'automatisation doit respecter des contraintes (ou échéances) temporelles dépendantes du processus contrôlé. Elle est donc suffisamment réactive si le délai de réaction est inférieur ou égal à une contrainte temporelle fixée. Sur la figure 1.6, l'émission de la sortie S répond à cette condition au contraire de l'émission de la sortie S'.

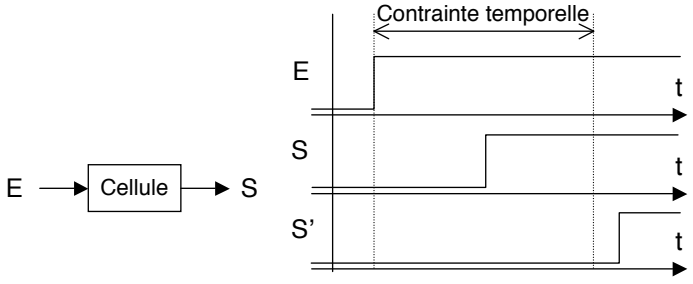


FIGURE 1.6 – Exemple illustrant le terme de réactivité

L'exemple précédent montre que la réactivité de la cellule va dépendre en partie de la capacité du réseau de terrain à offrir un délai de transmission inférieur à une contrainte temporelle donnée (délai de transmission maximum pour lequel la cellule peut être qualifiée

de réactive). Par analogie, nous appelons réactivité du réseau de terrain cette capacité.

Nous soulignons néanmoins que le respect d'une contrainte temporelle n'est pas une donnée suffisante pour conclure sur une caractéristique de réactivité, particulièrement dans le cadre de l'évaluation d'un réseau de terrain. En effet, la quantité de données à transmettre, la quantité de communications nécessaire comme les délais de propagation d'un signal sur le réseau physique vont conditionner la réactivité du réseau de terrain. Par conséquent, nous dirons qu'un réseau de terrain est réactif s'il respecte la contrainte temporelle dans des conditions d'utilisation fixées.

Deux types d'approches peuvent être utilisés pour mettre au point un système de contrôle réactif. L'auteur de [2] rappelle les deux types de paradigme qui peuvent être utilisés pour mettre au point un réseau de communication.

Un réseau *event-triggered* (ou *event-driven*) ne va être sollicité et ne va transmettre que des messages d'évènements, par exemple « la chaudière a atteint la température de consigne ». L'intérêt d'une telle approche est que l'utilisation du réseau est optimisée ; le réseau n'est pas utilisé inutilement lorsqu'aucun évènement d'entrée ou évènement interne (c'est-à-dire l'échéance d'une temporisation) ayant une sortie causale n'a lieu. La difficulté est d'être capable d'évaluer la robustesse de la caractéristique de réactivité d'un réseau *event-triggered* dans des conditions d'utilisation limites, c'est-à-dire en cas de nombreux évènements simultanés ou d'une cascade d'évènements. Un réseau *event-triggered* ne sera plus considéré comme assez robuste à un nombre important d'évènements rapprochés si tout ou partie d'entre eux est perdu (et non retransmis) ou si leur transmission est réalisée dans un délai supérieur à la contrainte temporelle fixée.

Dans un réseau *time-triggered* (ou *time-driven*) des messages d'état des entrées et des sorties vont cycliquement être transmis, même si celles-ci n'évoluent pas ou si l'évolution de leur valeur n'est pas significative. Par conséquent et par opposition à un réseau *event-triggered*, l'utilisation du réseau n'est pas optimisée. Par contre, l'avalanche de changements ne constitue pas une condition d'utilisation mettant potentiellement la réactivité du réseau en défaut. Ainsi, la contrainte temporelle de réactivité à respecter par le réseau pourra facilement être comparée à un multiple du temps de cycle [3] dans des conditions d'utilisation données.

Comme le note [4], systèmes réactifs et systèmes temps-réel sont souvent assimilés. À ce titre, [5] rappelle les différentes catégories de temps-réel :

- Temps Réel à Contraintes Strictes (TRCS), aussi appelé *temps-réel dur* ;
  les contraintes temporelles doivent être systématiquement respectées pour éviter des conséquences dangereuses.
- Temps Réel à Contraintes Relatives (TRCR), aussi appelé *temps-réel souple* ;
  des dépassements d'échéances temporelles peuvent être tolérés sans risque de provoquer des conséquences dangereuses.
- Temps Réel à Contraintes Mixtes (TRCM) ;
  des contraintes temporelles strictes et relatives sont demandées.

Nous utiliserons le terme *temps-réel* lorsque nous aurons besoin de nous référer à l'une de ces trois catégories.

Dans le cas d'un système de production d'énergie, l'élaboration des ordres pour le processus est majoritairement accompagnée d'exigences TRCS tandis que certaines demandes

de l'opérateur de supervision ne sont assujetties qu'à du TRCR. Le réseau de terrain étant impliqué dans toutes les communications, il doit permettre des communications TRCM, satisfaisant toutes les contraintes temporelles.

1.1.2 Exigence de déterminisme.

Dans [6], l'auteur définit qu'un système est déterministe si pour chaque état du système et chaque combinaison des entrées, il existe une seule combinaison des sorties et l'état suivant du système peut être déterminé. En particulier, on parlera de déterminisme causal si l'état suivant du système ainsi que la combinaison des sorties sont connus pour un état du système et une combinaison d'événements d'entrées. On remarque que le temps n'intervient pas dans cette définition.

On distingue le déterminisme causal du déterminisme temporel pour lequel le délai de réaction pour l'émission des combinaisons de sorties est connu quel que soient l'état et la combinaison des entrées. D'après cette définition, le déterminisme temporel est une exigence préalable lorsque l'on cherche du TRCS. Il faut que les délais soient connus et inférieurs ou égaux à la contrainte temporelle. Ainsi, un système sans déterminisme temporel sera qualifié de *best effort* [7]. Un système *best effort* ne pourra répondre qu'au mieux, c'est-à-dire sans garantie, aux contraintes temporelles fixées par le processus à contrôler.

Dans les applications de production d'énergie, impliquant des dispositifs de régulation, on recherche un déterminisme causal et temporel de la cellule d'automatisme. À la condition que le déterminisme temporel et la réactivité des communications entre programmes d'automatisation et processus soient assurés, le déterminisme causal est supporté par le programme d'automatisation. Ce dernier est exécuté par les contrôleurs. Par conséquent, le réseau de terrain doit répondre à une exigence de déterminisme temporel. Le délai de transmission d'une combinaison de trames doit être connu quelles que soient ces trames.

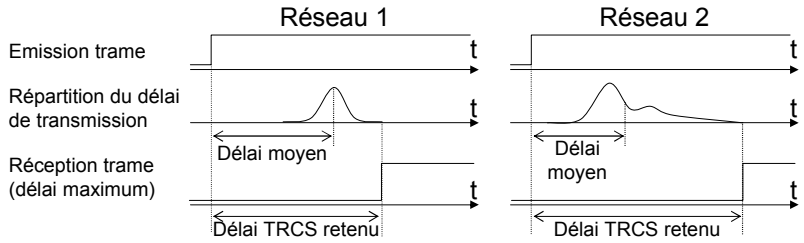


FIGURE 1.7 – Effet de la gigue sur le déterminisme

Notons que le délai de transmission est toujours accompagné de plus ou moins de gigue (variation du délai). Dans un contexte TRCS, le délai retenu ne pourra être que le pire délai possible. Si l'on compare les réseaux 1 et 2 de la figure 1.7, offrant des répartitions différentes de la probabilité de délai de transmission, le réseau 1 sera considéré comme plus performant du point de vue TRCS. D'un point de vue TRCR, le réseau 2 pourrait être considéré comme plus performant, son délai moyen étant plus faible.

1.1.3 Exigence de disponibilité.

Dans les processus critiques tels que ceux intervenant dans la production d'énergie, les différents attributs de la sûreté de fonctionnement doivent être respectés. Dans [8] les auteurs étendent les attributs classiquement donnés à la sûreté de fonctionnement. Du quatuor Fiabilité, Maintenabilité, Disponibilité (FMD), Sécurité (regroupé sous l'acronyme de Reliability, Availability, Maintenabiltity and Safety (RAMS) en anglais), ces attributs deviennent :

- la fiabilité, obtenue par la *continuité du service*,
- la sécurité innocuité, obtenue par l'*absence de conséquences catastrophiques pour l'environnement*,
- la confidentialité, obtenue par l'*absence de divulgations non autorisées de l'information*,
- l'intégrité, obtenue par l'*absence d'altération inappropriée de l'information*,
- la maintenabilité, obtenue par l'*aptitude aux réparations et aux évolutions*,
- la disponibilité, obtenue par le *fait d'être prêt à l'utilisation.*

Les différents éléments de la cellule d'automatisation doivent lui permettre de posséder ces différents attributs. En ce qui concerne le réseau de terrain, nous nous intéressons particulièrement à la disponibilité et en partie à la maintenabilité :

- conformément à la norme IEC61784-3 [74], l'intégrité sera obtenue par un protocole spécifique, indépendant du support de communication,
- la fiabilité, la sécurité innocuité et la confidentialité seront respectivement héritées des équipements utilisés, du programme de contrôle-commande et de la protection assurée au niveau supervision par le contrôleur de cellule.
- l'attribut disponibilité du réseau de terrain doit permettre à la cellule d'automatisation de respecter des exigences de réactivité et de déterminisme temporel en cas de défaillance d'un composant du réseau.

Plus précisément, nous considérons uniquement les défaillances par arrêt d'un composant du réseau, c'est-à-dire une suspension de leur service d'émission, réception ou transmission des communications, ayant pour conséquences des fautes du point de vue du réseau ou des autres composants interagissant avec lui (conformément à la chaîne fondamentale des entraves à la sûreté de fonctionnement [8] rappelée dans la figure 1.8). Le réseau de terrain disponible doit être tolérant à ces fautes afin que les erreurs alors générées ne soient pas propagées. Le but est d'éviter la propagation à la totale défaillance du service de communication rendu au sein de la cellule d'automatisation.

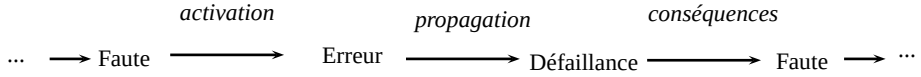


FIGURE 1.8 – Chaîne fondamentale des entraves à la sûreté de fonctionnement

En pratique, une meilleure disponibilité du réseau de terrain va le plus souvent être obtenue par redondance des composants logiciels et matériels susceptibles de provoquer les fautes par leur défaillance. La norme IEC61508-4 [75] définit la redondance comme « l'existence de plus de moyens que strictement nécessaire pour accomplir une fonction requise dans une unité fonctionnelle ou pour représenter des informations par des données ». Par conséquent, la défaillance d'un composant du réseau de terrain va se traduire pas une dégradation dans l'ensemble des services (ou fonctions) offerts (notamment, en supprimant

le service d'amélioration de la disponibilité) pour maintenir le service le plus important, à savoir le service de communication. En fonction de l'efficacité de la redondance, la propagation des erreurs générées peut :

- ne provoquer aucune défaillance. Les erreurs sont détectées et compensées avant qu'elles ne se propagent jusqu'au service de communication. La défaillance est totalement transparente pour la continuité du service de communication.
- provoquer une défaillance partielle entraînant une dégradation du service de communication. Dans le cas qui nous intéresse, nous considérons que la dégradation se traduit par une interruption momentanée du service (des pertes de trames, une augmentation de la contrainte temporelle de réactivité atteignable). Par exemple, nous ne considérons pas une dégradation se traduisant par une perte d'intégrité des données.

  Du point de vue des applications exigeant de la réactivité, l'interruption momentanée du service reste acceptable (et la défaillance reste partielle) tant que le retour à une pleine capacité du service n'est pas trop long, c'est-à-dire permet au processus de rester maîtrisable. La durée d'interruption du service autorisée sera d'autant plus faible que la contrainte de réactivité sera faible.

De plus, bien que l'exigence de disponibilité nous intéresse plus particulièrement, la détection et le signalement des erreurs causées sur le réseau de terrain sont également nécessaires. Ils vont permettre de lancer les actions de maintenance et retrouver le service d'amélioration de la disponibilité.

1.2 Objectifs des travaux

Nous venons de décrire la structure du système de contrôle-commande considéré et d'isoler la partie à laquelle nous nous intéressons exclusivement, le réseau de terrain de la cellule d'automatisation. Nous avons sélectionné trois exigences auxquelles doit répondre ce réseau de terrain dans des conditions d'utilisation définies :

- une exigence de réactivité. Le réseau doit satisfaire une contrainte temporelle de transmission des données afin que la cellule d'automatisation assure une réactivité de pilotage du processus.
- une exigence de déterminisme. Dans le but de répondre strictement à la contrainte de réactivité, le réseau doit satisfaire une contrainte de déterminisme temporelle afin que la cellule d'automatisation assure un déterminisme causal du pilotage du processus.
- une exigence de disponibilité. Un service d'amélioration de la disponibilité des communications doit être offert par le réseau afin que la fiabilité du pilotage du processus soit assurée.

Ces exigences sont issues d'une démarche plus générale de conception d'un système de contrôle-commande offrant un bon niveau de sûreté de fonctionnement. À ce titre, la norme IEC61508 recommande l'utilisation de différentes méthodes à appliquer à chaque étape du cycle de vie d'un tel produit. Ces méthodes témoignent qu'une démarche de réduction des risques a été suivie et sont indispensables pour des systèmes concernés par la sûreté de fonctionnement et candidats à une certification.

Étant donnée la criticité des processus destinés à utiliser le réseau de terrain, nous avons choisi de nous assurer que la spécification de la solution de réseau de terrain ne peut pas conduire le système dans un état non désiré. Un état non désiré sera un état pour lequel le système ne répond pas aux exigences fonctionnelles exprimées à partir des exigences

de réactivité, de déterminisme et de disponibilité. Parmi les méthodes recommandées par l'IEC61508, nous avons eu recours à la modélisation comportementale des services spécifiés et des exigences fonctionnelles. La modélisation nous permet de procéder à l'exploration systématique des états occupés pour une architecture réseau donnée grâce à un outil de model-checking.

Dans la prochaine section, le cahier des charges va nous permettre de préciser (c.-à-d. de chiffrer) la réactivité désirée pour le réseau de terrain ainsi que les conditions de respect de cette réactivité. Nous allons également préciser le déterminisme temporel recherché ainsi que les critères de disponibilité à satisfaire. En effet, si la même architecture de système de contrôle-commande et les mêmes catégories d'exigence peuvent s'appliquer à d'autres domaines que l'énergie (comme la production de biens par exemple), nous allons montrer que ce cahier des charges a la particularité de spécifier à la fois une contrainte temporelle de réactivité faible et des exigences de déterminisme et de disponibilité, susceptibles d'amoindrir la réactivité du réseau de terrain.

1.3 Cahier des charges de la cellule d'automatisation

À partir de sa connaissance du marché, la société Alstom Power met continuellement à jour une liste d'exigences que le système de contrôle-commande ALSPA CONTROPLANT aura à satisfaire. Ces exigences héritent des exigences courantes issues des derniers appels d'offres ainsi que d'une démarche d'anticipation des besoins futurs. Le cahier des charges détaillé ci-après correspond au sous-ensemble de ces exigences qui concerne le réseau de terrain.

Les applications d'automatisation pilotent des procédés nécessitant de plus en plus de précision, c'est-à-dire de gérer de plus en plus de paramètres tout en améliorant la réactivité de l'automatisme. Dans le domaine de l'énergie notamment, cette précision permet d'améliorer le rendement des processus pilotés.

En parallèle, les flux *orientés débit* augmentent également comme le note [9]. Depuis la salle de commande connectée au réseau de supervision, les exploitants d'une centrale veulent pouvoir observer l'évolution des variables ou l'état des équipements, mettre à jour les programmes ou les paramètres des équipements des cellules d'automatisation sans interrompre le pilotage du processus.

À l'instar du système de contrôle-commande présenté dans la section précédente, l'objectif du CONTROBLOC est d'avoir une architecture type de réseau de terrain qui puisse être appliquée sur différents processus de la production d'énergie, c'est-à-dire dans des conditions d'utilisation différentes. Par exemple, une cellule d'automatisation avec la même solution de réseau de terrain doit permettre le contrôle de machine tournante comme l'automatisme distribué d'un barrage hydroélectrique. Dans le cas du réseau de terrain, l'architecture type que nous considérons comprend l'ensemble de l'infrastructure matérielle (cartes de couplage réseau, équipements d'interconnexion, câbles), ainsi que les briques logicielles gérant les communications d'un protocole.

Le réseau de terrain doit permettre le respect de la contrainte de réactivité pour l'union des conditions imposées par différents processus. L'évolution de ces différents processus a conduit notamment Alstom Power à définir une contrainte de réactivité de 50ms au niveau cellule d'automatisation. Pour y répondre, les temps de traitement par un contrôleur et d'acquisition des entrées ou restitution des sorties sont fixés par les équipements.

Par conséquent, le réseau de terrain doit offrir une réactivité des échanges qui satisfasse à la contrainte de temps restant. Les conditions de réactivité fixées par le cahier des charges sont les suivantes :

- jusque 30 nœuds terminaux doivent pouvoir communiquer sur le réseau. Ces nœuds terminaux sont des contrôleurs et des équipements d'entrées sorties déportées.
- une quantité de données atteignant 10 kilo octets doit pouvoir être échangée entre les nœuds dans l'intervalle de temps correspondant à la contrainte temporelle de réactivité.
- le réseau doit pouvoir s'étendre jusqu'à une distance de 5 kilomètres (cas des barrages hydroélectriques).
- le réseau doit supporter des communications ayant une contrainte temporelle souple (essentiellement destinées à la supervision et à la configuration pas à pas) en parallèle du flux destiné aux échanges de variables de pilotage du processus.

Pour toutes ces conditions, le réseau de terrain doit répondre à une contrainte temporelle de réactivité dépendante de l'approche utilisée pour spécifier le réseau de terrain (*event-triggered* ou *time-triggered*). En effet, pour répondre à la contrainte temporelle, le temps de cycle des échanges d'une solution *time-triggered* devra être conforme au théorème de Shannon, c'est-à-dire au moins deux fois plus strict que pour une solution *event-triggered*.

Nous partageons l'avis de [3] qui remarque que la propriété de tolérance aux fautes recherchée donne une solution *time-triggered* comme plus naturelle qu'une solution *event-triggered*. Toutefois les auteurs de [10] ont montré qu'une solution *event-triggered* pouvait être envisagée.

Le cycle des échanges de variables applicatives offert par un réseau de terrain *time-triggered* doit respecter une contrainte temporelle de 5ms dans les conditions d'utilisation et pour les fonctionnalités définies. La contrainte temporelle de réactivité étant stricte pour les échanges de variables de pilotage du processus, le déterminisme temporel est requis. Enfin, afin que le processus puisse être piloté même en cas de défaillance sur le réseau de terrain (par exemple en cas de rupture d'un câble), le réseau de terrain doit offrir des fonctions de redondance d'équipement et de chemin entre nœuds terminaux. De la même manière, la fonction d'arbitrage des communications sur le réseau de terrain ne doit pas être supportée par un unique équipement afin que les communications (donc le pilotage) puissent perdurer malgré une défaillance de cet équipement.

WorldFIP est un des bus de terrain utilisés actuellement par le CONTROBLOC. Bien que sa technologie réponde intrinsèquement aux exigences de déterminisme temporel et de disponibilité, ses performances ne sont plus développées depuis plusieurs années. Par conséquent, les performances n'ont pas suivi l'augmentation des besoins en réactivité de ses applications.

De son côté, les performances du support Ethernet sont constamment améliorées. À tel point qu'il est déjà devenu une alternative et est perçu comme le successeur des bus de terrain classiques pour le contrôle-commande de processus industriels. La section suivante va donc traiter des solutions Ethernet industriel et dégager les conditions de respect du cahier des charges d'Alstom Power systems.

1.4 Les solutions Ethernet Industriel

L'Ethernet standardisé par l'IEEE dans la norme 802.3 et à l'international par la norme IEC8802-3, est un support développé à l'origine pour les applications bureautiques. Il couvre les couches 1 et 2 du modèle OSI (Figure 1.9), c'est-à-dire la façon dont les données sont transmises électriquement sur le médium physique et les procédures de transmission de données entre nœuds.

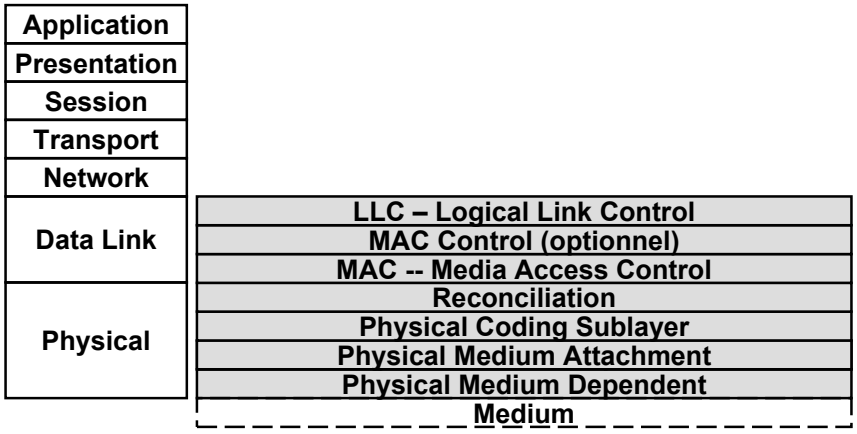


FIGURE 1.9 – Position d'Ethernet par rapport au modèle OSI.

Son utilisation a déjà dépassé le cadre de la bureautique pour certaines applications industrielles en remplacement des bus de terrain traditionnels tels que WorldFIP ou Profibus. En effet, l'alternative Ethernet propose des caractéristiques attrayantes :

- une augmentation constante des débits possibles. À l'heure actuelle des couplages jusque 1Gbit/s sont proposés,
- une taille de trame possible jusque 1526 octets (entête incluse),
- les coûts de développement sont supportés et amortis par les acteurs de la bureautique. Il en résulte une diminution du coût du support physique et éventuellement du couplage.
- par conséquent, l'offre est régulièrement enrichie par de nombreuses solutions basées sur du matériel standard, avec des possibilités de connexion simples ou durcies. Ces dernières sont développées afin d'être utilisées dans des environnements difficiles (par exemple en température ou humidité).

Ayant été supportés principalement par la bureautique, les choix technologiques qui ont conduit au développement d'Ethernet (et en particulier le développement du débit théorique possible) ne sont néanmoins pas toujours adaptés aux applications industrielles.

En premier lieu, Ethernet n'est plus un bus. Des équipements d'interconnexion (commutateurs, concentrateurs) sont nécessaires. Ces équipements nécessitent une alimentation qui n'est pas fournie par Ethernet et sont susceptibles de tomber en panne. En plus du surcoût par rapport à un simple câble, les équipements d'interconnexion se révèlent donc être des éléments critiques de la disponibilité du réseau.

La norme Power over Ethernet (PoE) IEEE802.3af permet d'acheminer une alimentation sur le même câble que celui de transmission des données. Toutefois, il est essentiellement destiné à l'alimentation des équipements terminaux, donc ne permet pas d'envisager une alimentation pour plusieurs équipements chaînés, d'autant plus que la puissance pos-

sible est faible (12,95W).

Ensuite, comme expliqué par [11], Ethernet n'est pas déterministe. La méthode d'accès au médium Carrier Sense Multiple Access - Collision Detection (CSMA-CD) est une méthode multi maître qui ne garantit ni le délai d'acheminement d'une trame sur le réseau, ni l'acheminement effectif de la trame. Dans [12], l'auteur explique que des propositions ont été faites très tôt pour bénéficier d'un Ethernet déterministe, mais qu'elles ne sont économiquement pas viables aujourd'hui.

Par contre, la généralisation des commutateurs dérivés de la norme IEEE 802.3D a permis d'utiliser les mécanismes du *Full-Duplex* et ainsi de donner plus de déterminisme à l'acheminement des trames. Alors qu'un concentrateur répète toutes les trames reçues sur ses autres ports en *Half-Duplex*, c'est-à-dire que le signal ne peut circuler que dans un seul sens dans un lien, sous peine de collision, le *Full-Duplex* permet à un concentrateur d'émettre et de recevoir sur un port en même temps. Les éventuelles concurrences d'arrivée de trames seront résolues dans le commutateur par des mécanismes de files d'attente. Ainsi, le délai d'acheminement des trames peut être borné. À ce titre, notons que [13] [14] et [15] proposent des modélisations du comportement des différentes technologies de concentrateur.

C'est pourquoi l'Ethernet commuté est présenté comme une solution de « réseau temps-réel et déterministe » [16] et des travaux tels que [17], [18] ont permis d'évaluer les contraintes temporelles et les conditions matérielles associées pour lesquelles Ethernet commuté peut être considéré comme réactif. Toutefois, les commutateurs (utilisant la technologie *Store-And-Forward*, seule technologie disponible aujourd'hui) introduisent :

- des délais pour l'acheminement des trames qui peuvent se révéler non négligeables en fonction de la contrainte temporelle de réactivité voulue. En effet, alors qu'il faut 0,56μs à un bit d'une trame Ethernet pour parcourir 100m de câble cuivre (0,5μs pour une fibre optique) et 1μs pour traverser un concentrateur, il lui faut de 12μs (pour une trame Ethernet de 64 octets) à 125μs (pour une trame Ethernet de 1526 Octets) en *Fast-Ethernet* (100Mb/s) pour traverser un commutateur *Store-And-Forward* (d'après les chiffres fournis par [76] et certains constructeurs d'équipements comme Hirschmann [77]).
- une gigue (variation autour du délai moyen) importante qui résulte de la politique de mise en file d'attente implémentée.
- un risque de congestion d'un équipement pouvant nécessiter une contention de certains flux. Si les commutateurs industriels actuels sont dimensionnés pour supporter une charge importante, comme montré dans [19], les couplages des nœuds terminaux peuvent être pris en défaut.

Par conséquent, s'il élargit l'espace des conditions de réactivité possible, Ethernet commuté n'apporte pas un gain significatif en terme de contrainte de réactivité par rapport aux bus de terrain traditionnels.

En pratique, les solutions industrielles basées sur Ethernet, Transmission Control Protocol (TCP) et Internet Protocol (IP) ne permettent pas de configurer une période de rafraîchissement (donc de garantir une contrainte temporelle de réactivité) inférieure à 10ms d'après les auteurs de [20].

Indépendamment de l'utilisation de commutateurs, les solutions universitaires telles que FTT-Ethernet [21], Rether [22], RT-EP [23], RT-net [24] ainsi que les solutions commerciales telles que EPL [78] [25], Profinet [79] [26] dans sa version Isochronous Real-Time (IRT) [27], EtherNet-Industrial Protocol (EtherNet-IP) + CIPsync [28], EtherCAT [80]

ou l'Avionics Full-Duplex Switched Ethernet (AFDX) ont donc proposé différentes optimisations pour permettre d'obtenir du déterminisme et une bonne réactivité avec Ethernet. En effet, il est possible de gagner du temps sur les communications :

- il est possible de réduire le délai introduit par les trames en augmentant le débit,
- il est possible de réduire le délai de propagation des trames en réduisant les temps de traversée des équipements d'interconnexion et, uniquement pour les équipements qui stockent les trames avant de les envoyer (par exemple les commutateurs *Store-And-Forward*) en augmentant le débit,
- il est possible de réduire le délai de réaction des nœuds. Cette réduction sera d'autant plus grande que peu de couches du modèle OSI seront impliquées (en effet la gestion d'une couche supplémentaire implique des traitements supplémentaires). Déjà les bus de terrain tels que WorldFIP ou profibus utilisaient un modèle de communication réduit à 3 couches [12]. De la même façon, les réseaux utilisant Ethernet, TCP et IP, comme Modbus-TCP, offrent moins de réactivité que les solutions utilisant une pile de communication optimisée sur Ethernet. Un nœud terminal aura également un délai de réaction beaucoup plus faible si les communications sont gérées de façon matérielle que si elles sont gérées de façon logicielle dans un environnement (système d'exploitation) Best-effort,
- il est possible d'optimiser les échanges au cours d'une période en minimisant le rapport $\frac{Nombredetramesparperiodedechanges}{Quantitededonneesparperiodedechanges}$. Par exemple, un mécanisme maître-esclave où un nœud esclave n'émet une trame que s'il y est invité par un nœud maître nécessite au moins deux trames pour que la donnée soit produite. Nous avons également déjà fait remarquer dans la sous-section 1.1.1 qu'un système *event-triggered* optimisait ce rapport à la différence d'un système *time-triggered*.

Les différentes solutions disponibles sur le marché ont été évaluées par le département de recherche et développement d'Alstom Power Control Systems en 2004. Le but était de choisir un réseau de terrain répondant aux critères énoncés dans la section 1.3. Ethernet PowerLink a été choisi pour différentes raisons :

- l'application dans l'avionique de l'Ethernet AFDX est très performante au regard de la réactivité, du déterminisme et de la redondance. Toutefois, les équipements d'interconnexion et les couplages ont été développés avec des contraintes spécifiques. Il en résulte que le tarif de cette solution ne veut et ne peut rivaliser avec les solutions candidates à la standardisation.
- parmi ces solutions, décrites dans [29], seuls EPL, Profinet IRT, EtherNet-IP + CIP-sync ou EtherCAT affichaient une réactivité permettant de satisfaire les contraintes temporelles dans les conditions de réactivité du cahier des charges, et le support de constructeurs d'équipements.
- EPL était la solution la plus en avance à cette date.
- la solution (qui sera développée dans le chapitre suivant) s'est révélée être une solution simple et adaptée à la variété de processus à piloter (les automatismes distribués), alors que les autres solutions sont optimisées pour des conditions de réactivité différentes du cadre fixé par le cahier des charges (par exemple, le contrôle d'axes).
- enfin l'Ethernet PowerLink Standardization Group (EPSG), chargé de la promotion de la solution EPL, affichait une ouverture à la proposition d'extensions des fonctionnalités de la solution.

Cette dernière raison s'est révélée dominante. En effet EPL, comme les solutions concurrentes, ne proposait aucune offre permettant de répondre à la contrainte de disponibilité du cahier des charges. La possibilité d'initier une réflexion et de participer à la spécification de cette fonctionnalité était donc pour Alstom Power une condition importante du choix d'une solution.

Dans la section suivante, nous allons voir les solutions de disponibilité développées pour l'Ethernet bureautique ainsi que les différentes solutions destinées à l'Ethernet industriel.

1.5 Disponibilité sur Ethernet Industriel

Tout comme les applications industrielles considérées, de nombreuses applications bureautiques ont besoin de mécanismes d'amélioration de la disponibilité sur Ethernet. Dès lors, des solutions ont été proposées.

- le *Rapid Spanning Tree Protocol* (IEEE 802.1w). Ce protocole permet à des commutateurs raccordés entre eux suivant un maillage de connaître l'état global du sous-réseau et d'aiguiller les trames sur un nouveau chemin d'accès au nœud destinataire si une défaillance est détectée en aval du chemin utilisé jusqu'alors.
- le *Link Aggregation* (aussi appelé *Port Trunking*) (IEEE 802.3ad). L'agrégation de liens permet de raccorder deux commutateurs ou un commutateur et un nœud terminal avec plusieurs liens. Ceci permet d'augmenter la bande passante, mais également d'avoir plusieurs liens de redondance entre les deux équipements. Toutefois, l'agrégation de lien ne résoud pas la défaillance du concentrateur donnant l'accès à un nœud terminal.
- des équipements propriétaires comme le Link-Protector© (Shure Microsystems). Cette solution se présente comme un équipement externe à trois ports pouvant raccorder un nœud terminal avec deux réseaux indépendants. L'équipement sélectionne un des deux réseaux en fonction de l'activité détectée sur chacun d'eux.

Toutes ces solutions ont été développées pour utiliser des constantes de temps appropriées à la bureautique sans considérer leur application à un réseau de terrain. Elles ne permettent pas le respect de la contrainte de temps de 5ms pour l'ensemble des échanges. Elles permettent une compensation d'une défaillance dans un délai compris entre 0,5 et 1 seconde.

Des solutions adaptées aux contraintes temporelles de l'Ethernet Industriel dérivées ou non des solutions bureautiques ont donc été proposées.

- la plupart des solutions proposées à l'heure actuelle sont propriétaires et utilisent une base d'anneau comme décrit dans [30]. Leur temps dit de *cicatrisation*, c'est-à-dire pour compenser la perte d'un lien, est plus faible que pour le *Rapid Spanning Tree*, 20ms pour la solution la plus réactive. Par contre, ces solutions sont incompatibles entre elles.
- trois solutions ont été proposées pour la standardisation dans l'IEC62439 [81]. En fait, chacune de ces solutions repose sur un protocole développé par leur contributeur. Ces trois protocoles se nomment *Media Redundancy Protocol* (proposé par Siemens sur le principe d'un anneau), *Parallel Redundancy Protocol* (proposé par ABB sur le principe d'une redondance de médium), *Cross networks Redundancy Protocol* (proposé par Fondation Fieldbus sur le principe d'une redondance de médium).

Bien que plus performantes que leur origine bureautique, ces solutions ne permettent pas un rétablissement assez rapide des communications pour respecter la contrainte de temps

de 5ms pour l'ensemble des échanges. Avec 20ms de rétablissement des communications donc au moins 4 périodes d'affilée à l'issue desquelles les données seraient considérées comme perdues, le système de contrôle commande va considérer qu'il ne pilote plus le processus, et peut commander l'arrêt ou la mise en position de repli de celui-ci.

Les protocoles Ethernet industriels les plus performants, cités dans la section 1.4, ne peuvent donc bénéficier d'aucune solution de disponibilité existante leur permettant de conserver le même niveau de réactivité tout en prenant en compte le risque de défaillance des communications.

En cas de reconfiguration de chemin, les solutions dont les performances reposent sur la synchronisation, EtherNet-IP + CIPsync (utilisant le protocole IEEE1588 [82], normalisé par l'IEEE61588 [83]) ou Profinet IRT (utilisant une version modifiée de l'IEEE61588) risquent d'être soumis à un délai de resynchronisation avant d'être sûrs que la date d'envoi d'une trame permette toujours à tous les nœuds destinataires de la recevoir dans l'intervalle de temps autorisé.

Les solutions EtherCAT ou Sercos III où les nœuds sont chainés et modifient à la volée une trame qui les traverse tous proposent de boucler les deux extrémités de la chaîne pour profiter d'un lien de redondance. Celui-ci permettrait de reconfigurer le réseau dès la détection d'inactivité électrique sur un lien. Elles proposent également la possibilité de redonder l'équipement maître (qui est l'initiateur des trames Ethernet partagées entre les équipements) pour faire face à la défaillance de celui-ci. Néanmoins, aucun développement intégrant les deux dispositifs d'amélioration de la disponibilité n'était prévu lorsque les choix technologiques ont été réalisés.

Dans la seconde partie du chapitre suivant, nous allons donc développer les principes retenus dans la spécification de la solution de haute disponibilité pour EPL. Cette spécification a été élaborée par un groupe de travail de l'EPSG animé par Alstom Power. Avant cela nous allons présenter la solution EPL et ses spécificités.

1.6 Conclusion sur le chapitre

Ce premier chapitre nous a permis de décrire le système de contrôle-commande qui a constitué le contexte de nos travaux. Nous avons mis en évidence les critères de sélection pour une nouvelle solution de réseau de terrain. À partir de ces critères, un cahier des charges évaluant les besoins pour un futur réseau de terrain a été proposé. Ce cahier des charges considère une gamme d'applications orientée vers la production d'énergie.

En partant du constat que les bus de terrain actuels ne pouvaient répondre à ce cahier des charges, nous avons recensé les différentes approches utilisant Ethernet comme base d'un nouveau réseau de terrain dit Ethernet industriel performant (suivant le critère de la réactivité et du déterminisme) et nous avons constaté que plusieurs solutions étaient compatibles avec le cahier des charges. Nous avons alors justifié le choix d'une de ces solutions au travers du critère de disponibilité.

Concernant le critère de disponibilité, nous avons également recensé les différentes propositions d'amélioration de la disponibilité existantes avant de conclure à leur incompatibilité avec les solutions Ethernet Industriel offrant les meilleures performances de réactivité et déterminisme. En effet, les Ethernets industriels les plus réactifs nécessitent des solutions de disponibilité adaptées.

Le chapitre suivant détaille la solution Ethernet industriel retenue, EPL, ainsi que la solution adaptée d'amélioration de la performance, son extension EPL-HA.

Chapitre 2

Détail de la solution technique

Ce chapitre décrit le protocole EPL chargé de répondre aux besoins exprimés de réactivité et de déterminisme. Il décrit également son extension EPL-HA. Cette extension spécifie de façon indépendante la redondance des chemins de communication entre nœuds et la redondance d'arbitrage des communications sur le réseau. Le but d'EPL-HA étant d'améliorer la disponibilité des communications, le chapitre se termine sur l'énonciation des exigences fonctionnelles que l'extension doit satisfaire.

Sommaire

2.1 Description du protocole Ethernet PowerLink

2.1.1 Présentation

EPL est une solution Ethernet Industriel développée et proposée dès 2001 par le constructeur d'équipements d'automatisme Bernecker & Rainer (B&R automation). En 2003, B&R automation décide d'ouvrir EPL aux autres constructeurs et confie la promotion et le développement d'EPL à l'EPSG. L'Ethernet PowerLink standartization Group est constitué de constructeurs, d'utilisateurs de solutions d'automatismes et d'organismes intéressés par EPL. Outre l'accès aux spécifications mises au point par l'EPSG, les membres de l'EPSG bénéficient de la possibilité de proposer des extensions afin de compléter les domaines d'applications de la solution EPL. L'intégralité des membres peut également siéger au groupe technique de l'EPSG pour voter les modifications et les extensions d'EPL. Depuis fin 2007, EPL est incluse dans les standards IEC61784-2 [84] et IEC61158-300 à IEC61158-600 [85].

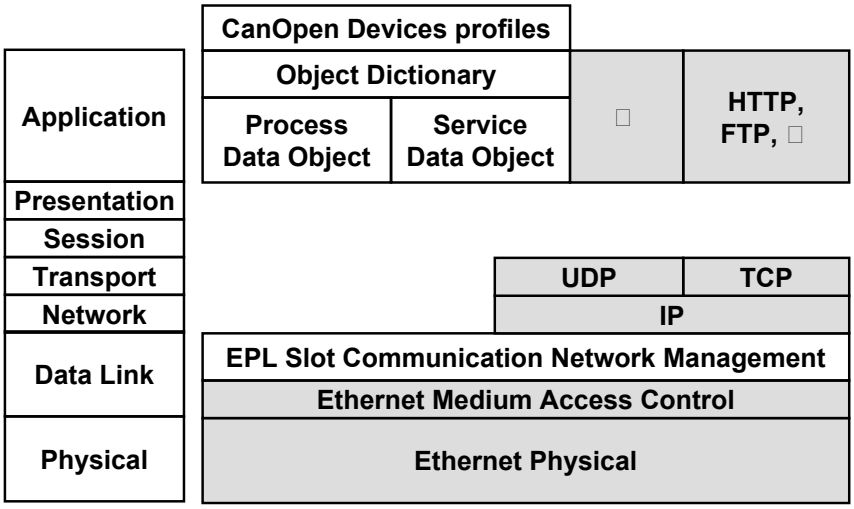


FIGURE 2.1 – Position des couches EPL par rapport au modèle OSI 7498

La figure 2.1 représente la solution EPL du point de vue du modèle OSI. EPL intervient à partir de la couche 2 du modèle. Cela signifie qu'aucune adaptation des couches servies par Ethernet n'est nécessaire pour implémenter EPL et que des composants Ethernet standard peuvent être utilisés. Ainsi, le protocole EPL n'a pas besoin d'être adapté pour les différentes évolutions d'Ethernet.

Les couches EPL peuvent donc êtres exécutées par un composant (Application-Specific Integrated Circuit (ASIC), Field-Programmable Gate Array (FPGA) ou processeur réservé) sur un coupleur dédié à EPL ou par l'intermédiaire du système d'exploitation sur le CPU de l'équipement avec une carte Ethernet standard. Bien sûr, la contrainte de réactivité supportée par un équipement sera moindre avec une gestion logicielle des communications qu'avec une gestion matérielle.

EPL permet d'obtenir des échanges déterministes en instaurant un mécanisme de communications qualifié de *publisher-subscriber pull-model* dans l'IEC61158 [85] et expliqué dans [12]. Ce mécanisme est géré par la couche EPL Slot Communication Network Management (SCNM) du protocole.

Les communications sur un réseau EPL sont régies par un unique nœud arbitre, le Managing Node (MN). Les nœuds esclaves, appelés Controlled Node (CN), d'une architecture produisent des trames Ethernet sur le réseau uniquement si ils y sont invités par le nœud MN. Ainsi, seule une trame sera envoyée à un moment donné sur le réseau et aucune collision ne peut intervenir. Le délai de transfert d'une trame d'un nœud à un autre ne dépend alors que des délais de traversée des composants d'interconnexion, et non des communications entre autres nœuds.

Par conséquent, il ne faut pas qu'un nœud non-EPL soit branché sur le réseau sous peine que les échanges soient perturbés et le déterminisme perdu. Un réseau EPL doit être déployé sur un sous-réseau isolé dédié à EPL. Plutôt que constituer une contrainte, le découpage en sous-réseaux correspond au découpage en cellules d'automatisation présentées dans la section 1.1.

La surcouche EPL prévenant les collisions, l'utilisation de commutateurs n'est pas nécessaire pour obtenir le déterminisme du temps de transmission entre les couches de niveau 2 du modèle OSI et améliorer la réactivité.

Les concentrateurs, comparés aux commutateurs, souffrent d'une image d'obsolescence héritée du point de vue bureautique. En effet, dans le contexte de la bureautique, il suffit que le réseau autorise un temps-réel souple (avec une faible contrainte temporelle de réactivité) ainsi qu'un débit important pour satisfaire l'utilisateur. Dans les applications multimédias, par exemple, la mise en mémoire tampon permet de compenser les écarts de régularité. La commutation a donc un réel avantage.

Par contre, dans un contexte industriel, les concentrateurs ont des atouts compétitifs (en terme de simplicité et de délai introduit) comparés aux commutateurs. C'est pourquoi EPL préconise l'utilisation de concentrateurs plutôt que de commutateurs en dépit de leur image techniquement obsolète.

Comme nous l'avons vu dans la section 1.4, les commutateurs introduisent un délai de traversée de 10 à 100 fois supérieur au délai de traversée d'un concentrateur. Dans le cas où tous les nœuds possèdent un commutateur intégré à leur carte de couplage réseau et sont connectés en ligne (*Daisy-Chain*), la somme de ces délais pourtant faibles peut avoir un poids important sur la réactivité. En l'absence de collisions, les concentrateurs participent donc à une meilleure réactivité du système.

Les commutateurs introduisent également une gigue plus importante que les concentrateurs. Alors qu'un concentrateur introduit une gigue de 40ns, c'est-à-dire que le délai de traversée peut varier de plus ou moins 40ns autour du délai moyen de traversée, un commutateur introduit une gigue de plusieures μs.

Ces différences de comportement temporel sont dues au niveau d'intervention des deux différents équipements. Le concentrateur intervient au niveau 1 du modèle OSI ; il recopie bit à bit les trames arrivant au niveau d'un port sur les autres ports. Le commutateur intervient au minimum au niveau 2 du modèle OSI ; l'adresse du ou des destinataires d'une trame arrivant à un port sera analysée pour que la trame ne soit recopiée que sur le ou les bons ports. Parmi les types de commutateurs recensés par [31], ce comportement correspond à la technologie *Cut-Through*, toutefois seuls des commutateurs *Store-And-Forward* sont commercialisés actuellement. Un commutateur *Store-And-Forward* analyse l'intégralité d'une trame arrivée sur un port afin de s'assurer de sa validité avant de la transférer. Ces différences de traitement expliquent les différences de performance temporelle entre concentrateur et commutateur.

Le comportement d'un concentrateur a pour résultat que toute trame envoyée sur le réseau est potentiellement reçue par tous les nœuds du réseau. Une nouvelle fois, cette conséquence apparaît comme une surcharge inutile des nœuds non concernés du point de vue bureautique, où les connexions sont essentiellement point à point. Par contre, cela permet à une application industrielle d'envisager la redondance d'équipements ou la surveillance de communications depuis n'importe quel point du réseau sans avoir à sélectionner spécifiquement des commutateurs gérant efficacement les communications en diffusion restreinte (*Multicast*) et le *Port Mirroring*. Notons également que tous les dispositifs de commutation implantés ne gèrent pas les communications en diffusion restreinte ou diffusion (*Broadcast*) avec la même efficacité temporelle comme le souligne les auteurs de [31].

Afin de limiter la latence introduite par les commutateurs, la solution Profinet IRT impose l'utilisation de commutateurs spéciaux fonctionnant en mode *Cut-Trough* lorsque des trames Profinet temps-réel doivent transiter, et en mode *Store-And-Forward* le reste du temps. En admettant que ces commutateurs soient utilisés avec un autre type de trafic temps-réel, comme EPL, le délai de traversée sera déjà 4 à 5 fois supérieur (à 100Mbits/s) au délai de traversée d'un concentrateur.

Un commutateur réalisant des fonctions plus évoluées utilise des programmes de traitement plus compliqués. Nous pensons que la probabilité qu'il souffre d'une défaillance est plus importante que dans le cas d'un concentrateur.

Enfin, la plus grande simplicité intrinsèque d'un concentrateur permet à un concentrateur durci d'être moins cher qu'un commutateur durci à nombre de ports égal. Les commutateurs d'entrée de gamme ont tendance à gagner en fonctionnalités à prix constant. Ils restent par conséquent toujours plus chers que des concentrateurs.

Néanmoins, les concentrateurs offrant moins de fonctionnalités qu'un commutateur, ce dernier conserve certains avantages pour une application industrielle. Par exemple, pour un commutateur fonctionnant en *Full-Duplex*, le débit théorique atteignable entre deux nœuds est de 200Mbit/s en *Fast-Ethernet*.

Les concentrateurs du marché ne gèrent pas le protocole Simple Network Management Protocol (SNMP). Ce protocole n'est pas géré par les commutateurs dits *non-manageables* non plus. Ces derniers n'interviennent que jusqu'au niveau 2 du modèle OSI. Les commutateurs *manageables* qui gèrent le protocole de niveau 7 (application) SNMP peuvent rendre compte de leur perception du réseau, ce qui peut permettre de superviser leur état ou de découvrir automatiquement l'architecture du réseau.

Un dernier inconvénient des concentrateurs est qu'ils ne sont pas destinés à une utilisation sur *Ethernet-Gigabit*. Le passage à un réseau EPL sur *Ethernet-Gigabit* s'accompagnera donc soit de l'utilisation de concentrateurs *Gigabit*, non commercialisés à l'heure actuelle, soit de l'utilisation de commutateurs *Gigabit* qui apporterait un gain faible par rapport à une solution EPL à base de concentrateurs en *Fast-Ethernet*.

Néanmoins, nous pensons que le passage à l'*Ethernet-Gigabit* n'est pas non plus exempt de difficultés pour les solutions ayant fait le choix de modifier les couches Ethernet jusqu'au niveau 1 (Physique). Pour ces dernières une reconception des composants (i.e. des ASIC, préférés aux FPGA jusqu'à présent par ces solutions) gérant le protocole sera nécessaire pour bénéficier de l'augmentation de débit d'une nouvelle couche physique.

2.1.2 Cycle des communications EPL

Comme l'illustre la figure 2.2, un cycle EPL élémentaire est constitué :

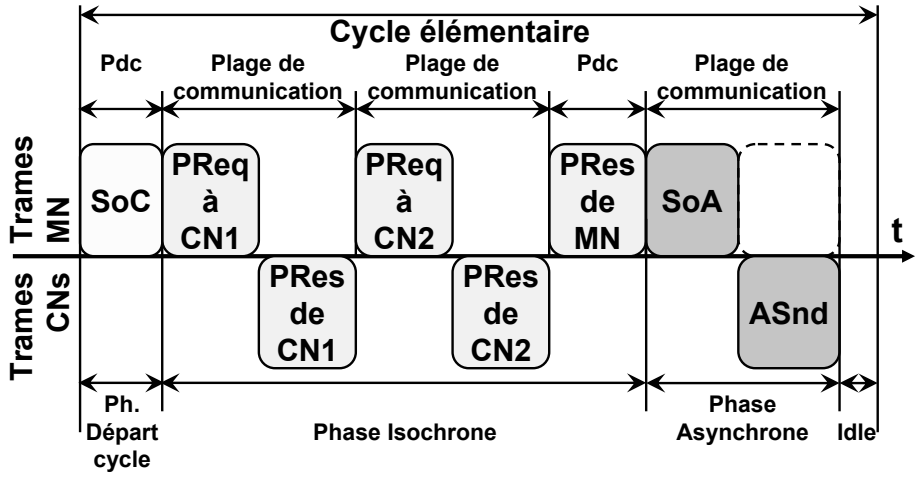


FIGURE 2.2 – Cycle élémentaire EPL avec deux CN.

D'une phase de départ cycle. Le Managing Node (MN) envoie une trame de signal de départ du cycle Start of Cycle (SoC) en diffusion restreinte (*Multicast*). Les dates d'envoi et de réception de cette trame servent de base commune pour la synchronisation. C'est pourquoi la trame SoC doit être générée périodiquement avec une bonne régularité.

D'une phase isochrone. Le MN envoie successivement une trame de commande de production des données applicatives Poll Request (PReq) à chaque Controlled Node (CN) configuré en point à point. Le nœud cible doit répondre en produisant une trame Poll Response (PRes) en diffusion restreinte. Une fois que tous les nœuds désirés ont produit leurs données, le MN peut envoyer sa trame PRes en diffusion restreinte.

Les données applicatives sont produites cycliquement par l'intermédiaire des trames PReq et PRes. Ainsi, conformément au modèle Producteur-Consommateur tous les nœuds ont la possibilité de consommer les données applicatives produites. Chaque nœud ne peut envoyer qu'une seule trame PRes dans la phase isochrone.

D'une phase asynchrone. Dans la phase asynchrone, le MN autorise un nœud contrôlé ou lui-même à transférer une et une seule trame dite asynchrone qui peut être une trame EPL Asynchronous Send (ASnd) (point à point ou diffusion restreinte) ou une trame Ethernet/User Datagram Protocol (UDP)/IP classique.

Le MN ouvre la phase asynchrone/clos la phase isochrone par une trame Start of Asynchronous (SoA) utilisée pour identifier les nœuds contrôlés inactifs, consulter les nœuds ne dialoguant pas dans la phase isochrone, vérifier l'état ou modifier la configuration des CN, ou autoriser un nœud à émettre une trame. Les CN réclament l'autorisation d'envoyer des trames asynchrones lors de l'envoi de leur PRes. Un mécanisme de files d'attente et de priorités est utilisé pour ordonnancer les différents messages asynchrones à autoriser.

D'une phase inactive. La phase d'inactivité est comprise entre la fin de la phase asynchrone et le début du cycle suivant. Sa durée dépend du temps pris par les communications précédentes et du temps de cycle élémentaire configuré. Sa durée minimale est de $0{,}96\mu s$ et correspond au temps inter trame (Inter Frame Gap (IFG)) obligatoire en *Fast-Ethernet*.

L'enchaînement phase de départ cycle → phase isochrone → phase asynchrone → phase inactive dans un cycle élémentaire est imposé. En revanche, les longueurs individuelles des différentes phases peuvent varier à l'intérieur d'un cycle tant que le temps de cycle élémentaire n'est pas dépassé.

Les caractéristiques du cycle EPL permettent d'évaluer l'adhésion de celui-ci à différentes combinaisons de conditions de réactivité définies dans la section 1.3. En effet, le déterminisme des échanges permet de calculer le temps de cycle nécessaire aux échanges entre un nombre donné de nœuds. Nous considérons un cycle élémentaire comme étant constitué de plages de communication comme illustré dans la figure 2.2. La figure 2.3 illustre les différents délais qui interviennent dans une plage de communication.

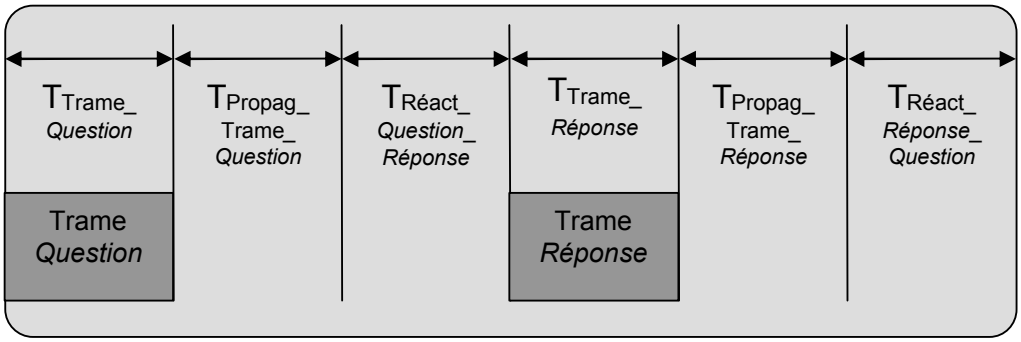


FIGURE 2.3 – Modèle de plage de communication EPL.

En appliquant le modèle de plage de communication aux échanges des différentes phases, il est possible de déterminer le temps minimum d'un cycle élémentaire EPL.

$$T_{\text{cycle elementaire minimum}} = T_{\text{Phase de depart cycle}} + T_{\text{Phase Isochrone}} + T_{\text{Phase asynchrone}} + T_{\text{Phase inactive minimum}}$$

Avec :

$T_{\text{Phase depart cycle}} = T_{\text{Trame SoC}} + T_{\text{React SoC PReq}}$

$T_{\text{Phase Isochrone}} = \sum(T_{\text{Trame PReq}} + T_{\text{Propag PReq}} + T_{\text{React PReq PRes}} + T_{\text{Trame PRes}} + T_{\text{Propag PRes}} + T_{\text{React PRes PReq}}) + T_{\text{Trame PRes}} + T_{\text{React PRes SoA}}$

$T_{\text{Phase asynchrone}} = T_{\text{Trame SoA}} + T_{\text{Propag SoA}} + T_{\text{React SoA ASnd}} + T_{\text{Trame ASnd}} + T_{\text{Propag ASnd}} + T_{\text{React ASnd SoC}}$

$T_{\text{Phase inactive minimum}} = 0,96\mu s$

Parmi les différents délais composant une plage de communication, leur donnée est obtenue à différents stades de la définition d'une application EPL.

Les délais maximaux de réaction des différents nœuds sont connus, car ce sont des caractéristiques intrinsèques des équipements (les valeurs sont fournies par le constructeur). Suivant le degré d'intégration du protocole EPL en gestion matérielle, de la puissance et des capacités de réactivité du couplage EPL, le délai de réactivité d'un nœud peut varier de $2\mu s$ à $50\mu s$.

Les durées des différentes trames sont connues dès lors que les quantités de données produites par chaque nœud sont définies par l'application. Les temps extrêmes d'une trame EPL sont déduits des tailles extrêmes de trame imposées par Ethernet et du débit considéré. À 100Mbit/s, compte tenu du fait qu'une trame Ethernet varie de 64 Octets à 1526 Octets, le temps d'une trame est compris entre 5,12μs et 122,08μs.

Les délais de propagation des trames sont connus dès lors que l'architecture physique du réseau à été fixée. Les durées associées aux trames et à la propagation influencent considérablement plus les performances avec un Ethernet industriel qu'avec un bus de terrain classique. Ces derniers utilisent souvent peu de composants d'interconnexion, et autorisent des trames plus petites (jusqu'à 256 octets contre 1526 pour Ethernet). La prise en compte de l'architecture physique est significative pour valider la réactivité d'une telle solution. La démarche des auteurs de [20] pour évaluer les performances des solutions Profinet IRT et EtherCAT confirme cette remarque.

L'évaluation analytique du temps de cycle EPL nous a permis de valider la réactivité théorique d'EPL pour une grande partie de l'espace des combinaisons de conditions de réactivité. De plus, EPL permet naturellement de répondre à une autre de ces conditions, la gestion de communications temps-réel souple, par l'intermédiaire de la phase asynchrone (présente à chaque cycle élémentaire). Néanmoins, les membres de l'EPSG ont recensé certaines fonctionnalités qui manquaient à EPL pour répondre non seulement au cahier des charges défini dans la section 1.3 mais également au cahier des charges d'autres applications. La sous-section suivante énumère ces différentes fonctions manquantes.

2.1.3 Fonctionnalités - Performances manquantes

Afin qu'EPL puisse être utilisé dans différents domaines d'application, l'EPSG a choisi d'enrichir le protocole existant à l'aide d'extensions qui ajoutent des fonctionnalités au protocole sans remettre en cause ses principes. Ceci doit permettre la compatibilité des équipements avec les spécifications d'origine, mais aussi d'adapter la complexité et le coût d'intégration d'EPL aux applications visées. On recense différentes extensions auxquelles réfléchissent les membres de l'EPSG.

Dans la sous-section 2.1.2, nous avons vu que le protocole permettait à un nœud d'envoyer un seul type PRes. De même, nous constatons que si le temps de cycle élémentaire configuré est grand comparé aux échanges, la phase d'inactivité sera plus longue sans que la phase des échanges asynchrones ne bénéficie du temps restant. Les membres de l'EPSG réunis dans le groupe de travail des extensions multimessages travaillent d'une part à augmenter le nombre de trames PRes possibles par nœuds, et d'autre part à améliorer le débit de trames asynchrones en utilisant la durée d'inactivité.

Comme souligné dans la sous-section 2.1.1, l'adoption de l'*Ethernet-Gigabit* par EPL soulève des interrogations quant à la conservation des concentrateurs. Le groupe de travail EPL-Gigabit de l'EPSG étudie l'évolution d'EPL vers de plus hauts débits.

L'extension EPLsafety spécifie des mécanismes permettant de s'assurer de l'intégrité des données échangées. EPLsafety spécifie un format de trame et des communications indépendants du support. Le format de la trame EPLsafety permet de diminuer la probabilité qu'une trame reçue et erronée (suite à les perturbations sur le chemin de communications) ne soit pas détectée comme telle. Bien que mis au point par les membres de l'EPSG, son utilisation ne se limite pas aux réseaux EPL. Les trames EPLsafety peuvent être encapsulées et transiter sur n'importe quel protocole ou architecture satisfaisant la contrainte de réactivité de l'application. Néanmoins, les premières implémentations de l'extension EPLsafety utilisent EPL comme réseau de communication support.

Enfin, le cahier des charges de la section 1.3 a souligné la nécessité d'une fonction de redondance des possibilités de communication. Entre 2005 et 2007, Alstom Power Control Systems a participé au groupe de travail EPL-HA afin de spécifier la redondance logicielle permettant d'avoir un nœud MN actif et animateur des échanges, ainsi que la redondance matérielle permettant d'avoir un chemin de communication valide entre deux nœuds en cas de défaillance d'un composant du réseau. Les deux sections suivantes décrivent les deux types de redondances spécifiées dans [86].

2.2 EPL-High Availability : redondance logicielle

Comme nous l'avons vu dans la sous-section 2.1.1 de présentation d'EPL, les communications sur le réseau sont régies par le Managing Node (MN). Si ce modèle de communication apporte du déterminisme, l'inconvénient est qu'en cas de défaillance du MN, les Controlled Node (CN) ne sont plus autorisés à produire leurs données sur le réseau et le processus ne peut plus être contrôlé. Ainsi le MN est un élément critique de la disponibilité des communications sur EPL. La partie de la spécification EPL-HA qui traite de la redondance logicielle décrit une extension du protocole permettant à une application EPL d'être tolérante à des défaillances de la fonction d'animation des échanges du MN.

À cette fin, le groupe de travail EPL-HA a défini un nouveau type de nœud pour remplacer le MN. Ce nœud est appelé Redundant Managing Node (RMN). Un réseau EPL utilisant la redondance logicielle sera donc composé de CN et d'un à plusieurs RMN. Par contre, la fonction d'animation des échanges ne peut toujours être portée que par un seul RMN à la fois, ce RMN est alors appelé Active Managing Node (AMN). Les autres RMN d'une cellule sont appelés Stand-by Managing Node (SMN) et se comportent comme des CN tant qu'aucune défaillance de l'AMN n'a été détectée.

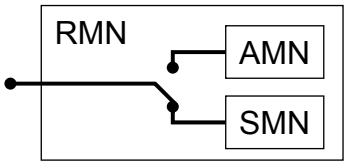


FIGURE 2.4 – Schéma d'un Redundant Managing Node.

En fonctionnement nominal, les SMN ne se distinguent donc des CN que parce qu'ils observent toutes les communications sur le réseau. Ainsi, ils sont capables de basculer à l'état AMN après observation d'un temps d'inactivité sur le réseau trop important (paramètre de configuration du nœud).

L'activité sur le réseau correspond pour un SMN à la présence de trames SoC et SoA (car elles sont produites exclusivement par l'AMN). À chaque RMN du réseau correspond un délai de basculement qui doit être unique pour garantir la présence d'un seul AMN à la fois. L'unicité du temps de basculement permet de fixer la priorité entre les RMN, le plus prioritaire ayant le temps de basculement le plus faible. La redondance d'arbitre du bus de terrain WorldFIP repose sur le même principe d'unicité et priorité de basculement.

Ce mécanisme de redondance de MN explique l'intérêt de la diffusion (ou au moins de la diffusion multipoint) des trames sur le réseau pour que tous les SMN soient capables d'observer les échanges et détecter une défaillance de l'AMN.

Afin de s'assurer que la redondance logicielle n'entraîne pas la présence de deux AMN sur le même réseau EPL, l'unicité du temps de basculement ne suffit pas. En effet, par le biais des temps de propagation des trames sur le réseau et de leur variation (gigue), il ne faut pas qu'un basculement inapproprié ait lieu à cause d'un délai de basculement trop court.

Il ne faut pas non plus que la différence de priorité soit trop faible entre deux RMN, c'est à dire inférieure ou égale à [(délai de propagation) + (gigue)] sous peine que le moins prioritaire bascule avant d'avoir reçu un signal de basculement du plus prioritaire.

Enfin, il ne faut pas qu'un basculement inadapté intervienne à cause d'une rupture du médium coupant le réseau en deux et dont le délai de réparation serait suffisamment grand pour qu'à la réparation de celui-ci, le réseau se retrouve avec deux AMN et que des collisions se produisent. C'est pourquoi, dans le but d'améliorer totalement la disponibilité et assurer la continuité des communications, la redondance logicielle doit pouvoir s'accompagner d'une redondance du support de communication telle que celle que nous détaillons dans la section suivante.

2.3 EPL-High Availability : redondance matérielle

Si la redondance logicielle permet d'améliorer la disponibilité de l'accès au médium par les nœuds, la partie redondance matérielle de l'extension EPL-HA permet d'améliorer la disponibilité de l'acheminement des trames. Si un câble venait à être sectionné ou si un équipement d'interconnexion (concentrateur, commutateur, convertisseur de média cuivre/fibre optique) venait à tomber en panne entre deux nœuds, il faut qu'un chemin alternatif puisse être utilisé pour maintenir les communications entre ces deux nœuds. Nous avons vu dans la partie 1.5 que les solutions industrielles du marché reposent essentiellement sur des architectures de type anneau. Ces architectures utilisent un lien de redondance qui est activé sur détection d'une coupure dans l'anneau.

Les dispositifs de redondance activés sur détection d'une défaillance (tel que la redondance logicielle décrite dans la section précédente) sont qualifiés de redondance passive par l'IEC67078 [87]. La détection implique un délai entre l'occurrence du défaut et l'activation du chemin alternatif (la cicatrisation, dans le cas d'un anneau) au cours duquel l'acheminement des trames n'est plus assuré. Par opposition, dans les dispositifs de redondance active (tel que décrit dans [32]), tous les chemins sont continuellement utilisés entre deux nœuds. Une défaillance n'entraîne donc aucun délai de perte des communications pour détection et reconfiguration des chemins.

Nous appelons redondance matérielle la partie de l'extension EPL-HA traitant de l'acheminement des trames, car cette partie repose sur la redondance du médium, c'est-à-dire de l'infrastructure réseau.

Les nœuds EPL communiquent sur deux réseaux indépendants sur lesquels les mêmes trames circulent en parallèle tant qu'aucune défaillance n'a lieu. L'utilisation d'un modèle de redondance active impose d'ajouter aux nœuds des mécanismes de filtrage afin de ne prendre en compte qu'une seule des trames identiques.

C'est pourquoi en plus de la redondance médium, le groupe de travail EPL-HA a spécifié une fonction complémentaire de la redondance de médium qu'elle a appelée Link Selector (LS). Comme l'illustre la figure 2.5, la fonction LS va s'intercaler entre les deux réseaux et un nœud, en interne ou externe (via un équipement supplémentaire).

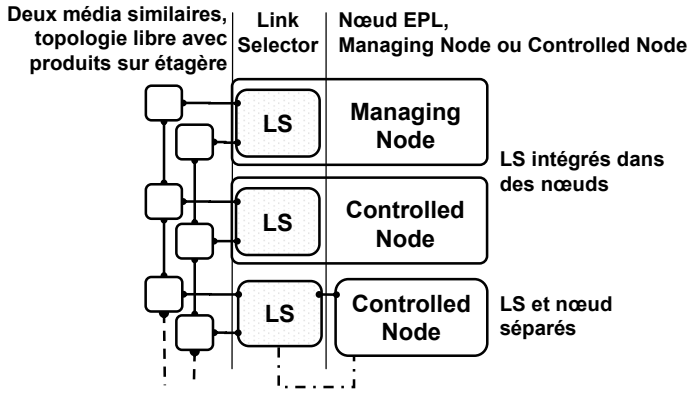


FIGURE 2.5 – Position de la fonction Link Selector dans l'architecture EPL-HA.

La figure 2.6 schématise les deux opérations réalisées sur les trames en fonction de leur provenance :

- la duplication des trames à l'émission,
- la sélection et le transfert de trame à la réception. La méthode de sélection entre les trames redondantes n'est pas imposée par la spécification. La méthode de sélection la plus simple et qui aurait le coût de traitement le plus faible serait de transférer la trame du médium le plus rapide une fois que la trame du médium le moins rapide est arrivée. Un délai d'attente maximum entre trames redondantes permettrait de détecter un problème sur un médium. Ce délai doit être suffisamment faible pour ne pas attendre trop longtemps et dégrader le délai de réaction du nœud. Il doit être également suffisamment important pour qu'un médium juste en retard soit détecté à tort comme ayant un problème. Ce délai doit être supérieur à ((délai le plus court sur le médium le plus rapide)-(délai le plus long sur le médium le moins rapide)). Des mécanismes peuvent être ajoutés comme la vérification de l'intégrité de la trame (vérification Frame Check Sequence (FCS)) au prix d'un temps de traitement plus long donc d'un allongement du délai de réaction du nœud.

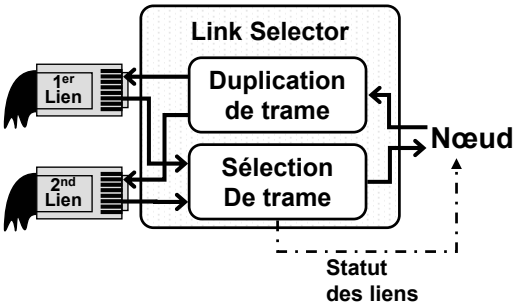


FIGURE 2.6 – Détail des opérations réalisées par la fonction Link Selector.

La redondance n'a de sens que si les défaillances peuvent être détectées et l'information remontée pour déclencher des actions de maintenance. La disponibilité du réseau de terrain peut alors être conservée. Afin qu'une défaillance soit détectée et localisée, chaque nœud EPL va renseigner des champs de ses trames PRes pour donner son point de vue de l'activité de chaque médium.

Depuis la supervision, le recoupement des différents points de vue de l'état de chaque médium permet d'isoler la zone de défaillance suspectée.

2.4 Exigences fonctionnelles

L'insertion d'une couche de disponibilité dans le modèle de communication EPL n'est pas exempt de conséquences. Nous voulons vérifier qu'au niveau de la spécification, la solution de redondance n'entraîne pas des effets de bord qui soient néfastes pour le déroulement des communications, ou simplement que la solution couvre bien les possibilités de défaillance et ne peut pas être prise en défaut. Pour cela, nous souhaitons valider la solution EPL-HA par l'intermédiaire d'exigences fonctionnelles qui devront être respectées. Nous avons retenu les exigences fonctionnelles suivantes qui, selon nous, permettent d'affirmer que le déroulement des communications n'est pas perturbé.

Il ne faut pas que deux SMN se comportent en tant qu'AMN en même tant dans la cellule d'automatisation. Il en résulterait très certainement des collisions de trames sur le réseau et l'activation du mécanisme CSMA-CD.

Or, si le modèle de communication utilisé par EPL prévient les collisions, il est possible que les dispositifs proposés fassent perdre cette propriété. Nous venons de décrire les cas occasionnés, par exemple, par un mauvais réglage des priorités de la redondance logicielle. En ce qui concerne la redondance matérielle, des collisions pourraient être engendrées par la fonction de duplication de trame.

Les mécanismes de redondance logicielle et matérielle ne doivent pas faire perdre la réactivité attendue pour le réseau. Il ne faut pas qu'un cycle élémentaire empiète sur le cycle suivant à cause de l'activation d'un mécanisme de redondance.

L'activation des mécanismes de redondance ne doit pas provoquer de détection de perte de trame du point de vue des nœuds EPL. Il faut également que ces dispositifs aident la supervision en remontant toute détection de défaillance pour déclencher des actions de maintenance curative. Il faut aussi qu'ils ne provoquent pas de détection erronée de défaillance qui pourrait se traduire par une position de repli du processus interrompant la production.

Ces exigences fonctionnelles traduisent les exigences au niveau du réseau de terrain pour que la sûreté de fonctionnement du processus piloté soit assurée. Nous allons utiliser la vérification formelle par model-checking, une des méthodes préconisées par l'IEC61508, pour valider la solution. L'utilisation de cette méthode implique une modélisation formelle des mécanismes étudiés. Par conséquent, nous avons défini d'une part un modèle paramétré d'architecture utilisant EPL-HA (les paramètres à renseigner permettent de décrire toute architecture particulière qui sera vérifiée), d'autre part nous avons modélisé les exigences fonctionnelles formulées ci-dessus sous forme de propriétés énoncées dans une logique temporelle.

2.5 Conclusion sur le chapitre

Dans ce chapitre, nous avons décrit le modèle de communication d'EPL producteur-consommateur *pull-model*, également appelé *producteur-consommateur-distributeur* par [33] utilisé par EPL et le déroulement des communications avec ce protocole. Nous avons pu constater que bien que répondant à la contrainte de réactivité fixée dans le chapitre 1 pour un domaine de conditions couvrant une grande partie du domaine défini par le cahier des charges, il manquait à EPL la fonctionnalité améliorant la disponibilité des communications. Après avoir énuméré les travaux en cours au sein de l'EPSG pour améliorer le protocole à partir d'extensions, nous avons pu détailler les deux types de redondance spécifiées par l'extension EPL-HA.

La redondance logicielle définit l'extension du protocole permettant la redondance de la fonction d'animation des échanges en spécifiant un nouveau type de nœud capable d'héberger une redondance passive de cette fonction.

Nous avons souligné à cette occasion le risque d'avoir plusieurs AMN sur un même réseau. Elle se traduirait par des collisions et l'indisponibilité du service de communications.

La redondance matérielle définit l'extension du protocole permettant la redondance de chemin de transmission des trames. Elle se traduit par une redondance de médium. Les communications sont réalisées indépendamment sur chaque médium. L'utilisation d'un médium redondant s'accompagne de la définition de la fonction LS chargée de réaliser la duplication des trames sortantes sur le réseau et la sélection des trames entrantes dans le nœud.

Le choix de ces deux composantes de redondance a pour but d'apporter des propriétés de disponibilités au protocole sans faire perdre les propriétés de réactivité et de déterminisme déjà acquises. Nous avons établi une liste d'exigences fonctionnelles nécessaires pour atteindre ce but. Nous désirons maintenant valider ces exigences par vérification formelle.

Le chapitre suivant est consacré à la description de la technique et de l'outil de vérification formelle choisis ainsi qu'à la démarche de modélisation que nous avons suivie, aussi bien pour le système à vérifier que pour les propriétés sélectionnées.

Chapitre 3

Modélisation

Les modélisations de la solution spécifiée et des exigences fonctionnelles décrites précédemment sont traitées dans ce chapitre.
Le but est de soumettre les modèles obtenus à un outil de model-checking pour pouvoir conclure sur leur adéquation. La syntaxe de l'outil de model-checking utilisé (Uppaal), les abstractions mises en œuvre et la démarche d'instanciation des modèles sont également détaillées.

Sommaire

3.1 Technique de vérification et modélisation choisies

Nous désirons vérifier des exigences traduisant la conservation de la réactivité et du déterminisme et d'obtention de la disponibilité sur un réseau de terrain utilisant EPL ainsi que l'extension EPL-HA. L'utilisation de ce réseau de terrain dans un contexte de production d'énergie lui confère une certaine criticité. Le système de contrôle-commande et son développement peuvent être audités pour l'obtention d'une certification à un niveau de sûreté. En effet, le système de contrôle-commande devra justifier d'une certification Safety Integrity Level (SIL) 1 à 4 suivant le processus à piloter. Ces SIL et leurs prérequis sont définis dans la norme IEC61508 [75].

Au titre de composant du système de contrôle-commande, la solution de réseau de terrain supporte une part des exigences permettant au système d'être certifié. Ces exigences concernent le processus de développement comme la solution de disponibilité apportée par le réseau de terrain. Une solution est apportée par la spécification EPL-HA. Nos travaux participent au processus de développement. Nous voulons nous assurer que la solution spécifiée répond aux attentes formulées sous forme d'exigences fonctionnelles (cf. section 2.4). Le but est de démontrer que la spécification n'autorise pas certains états du réseau conduisant à une perte de la sûreté de fonctionnement du processus.

3.1.1 Techniques de vérification

La sûreté de fonctionnement du processus repose sur la disponibilité des échanges qui doit être assurée par l'extension EPL-HA compte tenu de la réactivité et de la configuration des composants. Parmi les travaux dont l'objectif est de vérifier les conditions de temps-réel offertes par un réseau de terrain Ethernet, on note que beaucoup s'intéressent particulièrement aux commutateurs, identifiés comme les sources principales de dégradation des conditions. La modélisation sous des logiciels spécifiques (opnet [34] et [16]), l'utilisation du network calculus [14] ou d'autres modélisations analytiques comme [13] ainsi que des résultats expérimentaux permettent de définir des domaines d'utilisation.

Ces travaux s'intéressent à des communications sur Ethernet, TCP ou UDP, et IP donc à des réseaux TRCR. Même la modélisation et la simulation d'EPL par les auteurs de [25] s'intéresse plus particulièrement à la phase asynchrone et le temps-réel relatif qu'elle offre. Afin d'être capable de répondre à l'exigence de déterminisme temporel qui a été fixée (TRCS), nous avons décidé de recourir à une technique de vérification formelle, à l'instar des auteurs de [35].

Les techniques de vérification formelle sont issues de l'informatique. Dans [36] l'auteur souligne que leur utilisation est adaptée et appliquée aux systèmes à événements discrets depuis une dizaine d'années avec, en particulier, les travaux de Moon [37].

Les auteurs de [38] dressent une liste élargie de méthodes de vérification formelle : la simulation, le theorem proving [39], l'analyse d'atteignabilité et le model-checking [40].

La simulation ne permet pas de vérifier exhaustivement les états atteignables par le système. Un système simulable permettra juste de conclure qu'aucun état non désiré n'a été atteint au cours de la simulation. Il ne nous a pas paru adapté à notre problématique. Même si elle fait appel à des modélisations formelles, son statut de méthode formelle est d'ailleurs, à notre avis justement, discutée par de nombreux auteurs. Dans [41], par exemple, la simulation est présentée comme une technique de vérification distincte des méthodes formelles.

Le theorem proving consiste à déduire des propriétés sur le comportement d'un logiciel, un algorithme ou un protocole par l'intermédiaire d'un logiciel assistant de preuve (par exemple COQ [88]). Pour ce faire, l'assistant de preuve requiert une modélisation mathématique du sujet étudié, un protocole dans le cas qui nous intéresse, de son environnement et des propriétés. Les propriétés sont alors considérées comme les énoncés d'un théorème à démontrer à partir des modèles mathématiques du protocole et de son environnement pris comme des axiomes. Cependant, l'auteur de [42] rappelle que cette technique ne donne aucune garantie de résultat. De plus, elle peut nécessiter une intervention humaine experte pour réaliser la preuve de lemmes intermédiaires [43].

Le model-checking consiste à déduire des propriétés sur le comportement d'un logiciel, un algorithme ou un protocole par l'intermédiaire d'un moteur d'exploration exhaustive de l'espace d'état d'une modélisation. Pour cela, on utilise une représentation symbolique du modèle à vérifier qui ne modifie pas le comportement du modèle (par exemple, les BDD [44]) et des méthodes de réduction issues de l'informatique. Un schéma du principe du model-checking est donné par la figure 3.1.

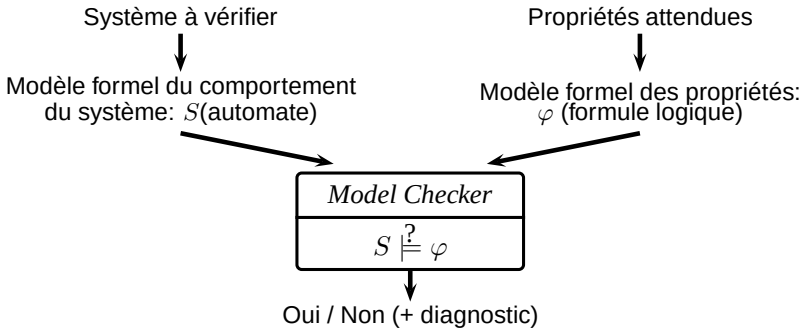


FIGURE 3.1 – Schéma général du principe du model-checking

On fournit par exemple à l'outil de model-checking une modélisation sous forme d'un système de transitions étiquetées S du sujet considéré. La modélisation doit être adaptée au moteur de model-checking choisi (automate à états finis, automate temporisé, réseau de Petri ...). Il faut également fournir une modélisation des propriétés à vérifier φ, par exemple sous forme de formules logiques utilisant la logique Computation Tree Logic (CTL) [45]. Le moteur va alors procéder à une analyse exhaustive de l'espace des états atteignables par le modèle S afin de déterminer s'il existe un chemin entre l'état initial du modèle et les états satisfaisant les propriétés φ. On note $S \models \varphi$, ce qui signifie que le modèle du système satisfait l'ensemble des propriétés φ. Si tel n'est pas le cas, l'outil de model-checking propose un contre exemple pour lequel les propriétés ne sont pas vérifiées par S.

L'inconvénient majeur du model-checking est l'explosion du nombre d'états à parcourir pour vérifier les propriétés, même si la représentation symbolique permet de diminuer le temps et la mémoire nécessaires à la vérification. Sur des modèles trop complexes, la construction et l'exploration de l'espace des états peut nécessiter plus que la quantité de mémoire maximum offerte par un ordinateur (4Go pour une application 32bits), et un temps de calcul conséquent.

Afin de réduire le nombre d'états à parcourir, on essaie alors d'abstraire le modèle du système étudié pour la partie du comportement n'ayant pas d'influence sur les propriétés

à vérifier. Toutefois, le risque de ne pas préserver ces propriétés est amplifié par rapport à une modélisation sans recherche d'abstraction. Néanmoins, de nombreux travaux ont utilisé avec succès des techniques de model-checking pour vérifier différents aspects de protocoles : [35] évalue des propriétés temps-réel du réseau de terrain sur Ethernet Modbus-TCP avec IO Scanning©(Schneider Electric), [46] vérifie des propriétés de l'IEEE802.15.4 [89] qui définit les couches basses utilisées par le protocole Zigbee.

Le besoin de sûreté de fonctionnement ne peut se contenter qu'une solution se comporte comme prévu la plupart du temps. Au contraire, l'objectif est de vérifier la continuité du bon fonctionnement dans les états les plus particuliers. À ce titre, le model-checking est hautement recommandé par l'IEC61508 pour les plus hauts niveaux de sûreté de fonctionnement. Le model-checking est d'ailleurs déjà utilisé chez Alstom Power pour la vérification formelle de contrôleurs logiques industriels [47]. Le model-checking est donc tout à fait adapté, compte tenu du contexte des travaux, pour vérifier formellement les propriétés de vivacité, de redondance et de déterminisme offertes par EPL avec son extension EPL-HA.

3.1.2 Outil de modélisation et de vérification

Il est possible d'obtenir des preuves, donc d'atteindre des objectifs de vérification avec n'importe quel moteur de model-checking. Toutefois, ces objectifs seront limités par la taille et le type du problème, pour lesquels le moteur ne sera pas forcément adapté. Nous avons donc cherché un outil de model-checking adapté à notre problématique.

Le réseau de terrain auquel nous nous intéressons relie différents composants dont le comportement individuel ainsi que les interactions sont fortement liés au temps. En effet, les changements individuels d'état seront provoqués par l'échéance de temporisations internes ou par la réception de certains messages. Chaque sous-système est donc à la fois *event-driven* et *time-driven*. De plus, nous avons vu dans la section 2.1.2 que le délai de transmission des messages n'est pas négligeable. Nous avons donc choisi d'utiliser un outil de model-checking capable de gérer du temps. Les environnements adaptés utilisent des techniques de model-checking temporel (Uppaal [48], KRONOS [49], TSMV [50], TReX [51]). Les environnements de model-checking hybride (HyTech [52], PHAVer [53], SPHIN[54]) ou de model-checking probabiliste (PRISM [55], APMC [43][56]) permettent d'aborder des problèmes où le temps n'est pas la seule caractéristique. Une conséquence de cette ouverture à une plus grande variété de problèmes est qu'ils ne sont pas les plus adaptés pour vérifier un modèle uniquement temporel.

Nous avons fait le choix d'Uppaal pour procéder à la vérification des propriétés recherchées. Les développeurs d'Uppaal définissent ce dernier comme un environnement de modélisation, de validation et de vérification de systèmes temps-réel. Dans le cadre de ces travaux, les interfaces de modélisation et de validation du modèle par la simulation offerts par l'environnement ont facilité respectivement la communication et la mise au point du modèle. Cet environnement est développé depuis une dizaine d'années par l'université d'Uppsala en Suède et l'université d'Aalborg au Danemark [90]. Uppaal a pour avantage d'être maintenu et de proposer régulièrement des améliorations de performances et de nouvelles fonctionnalités depuis sa première version sortie en 1999. Une conséquence que nous avons constatée à l'instar de [57], est qu'il fait souvent preuve de plus de robustesse que ses homologues pour valider des modèles conséquents. Il a d'ailleurs été utilisé pour la vérification de protocoles, par exemple dans un protocole de multicast dans [58] ou la vérification d'un protocole de bus de terrain industriel dans [59].

La modélisation sous UPPAAL est réalisée sous forme d'une variante des automates temporisés introduits dans les années 90 par Alur et Dill [60]. Dans la section suivante, nous allons rappeler la définition formelle des automates temporisés.

3.1.3 Définition formelle de la modélisation par automates temporisés

Les automates temporisés sont définis par Alur comme des automates à états de contrôle finis classiques munis d'un ensemble fini de variables appelées horloges comme le rappelle [61]. Ces horloges évoluent de manière continue et synchrone avec le temps. Elles sont utilisées pour spécifier des contraintes liées au temps sur les transitions ou sur les états de contrôle. Chaque transition contient une garde (condition sur la valeur des horloges) décrivant quand la transition peut être franchie et un ensemble d'horloges qui doivent être remises à zéro lors du franchissement de la transition. Chaque état de contrôle contient un invariant (condition sur la valeur des horloges) qui peut restreindre le temps d'attente dans l'état et par conséquent forcer le franchissement d'une transition et l'exécution de l'action associée.

Un exemple d'automate temporisé est donné dans la figure 3.2. Il modélise le fonctionnement d'un éclairage temporisé restant allumé 100 unités de temps après réception de la commande poussoir.

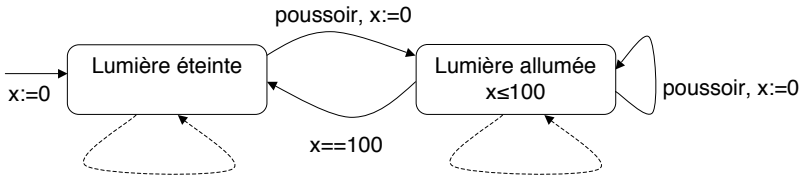


FIGURE 3.2 – Exemple d'automate temporisé modélisant un éclairage

Le domaine de temps peut être $\mathbb{N}$ (l'ensemble des entiers naturels), $\mathbb{Q}_{\geq 0}$ (l'ensemble des rationnels positifs), ou bien même $\mathbb{R}_{\geq 0}$ (l'ensemble des réels positifs).

X est un un ensemble d'horloges. Une valuation v des horloges de X est une application qui associe une valeur dans le domaine de temps à une horloge de X. $\mathcal{P}(X)$ est l'ensemble de toutes les valuations des horloges de X. $\mathcal{C}(X)$ est l'ensemble des condition sur les horloges X de la forme $x \bowtie c$ pour une horloge x, une constante c et l'opérateur $\bowtie \in \{<, \leq, =, \geq, >\}$.

Dans [62], les auteurs donnent les définitions formelles nécessaires à la description des automates temporisés. Un automate temporisé $\mathcal{A}$ est un 6-uplet $\langle X, \mathcal{Q}, q_0, \Sigma, E, \mathsf{Inv}\rangle$ où

- X un ensemble fini d'horloges,
- $\mathcal{Q}$ est un ensemble d'états de contrôle (ou de situations de contrôle),
- $q_0 \in \mathcal{Q}$ est l'état de contrôle initial,
- Σ est un alphabet qui désigne un ensemble fini d'actions,
- $E \subseteq \mathcal{Q} \times \mathcal{C}(X) \times \Sigma \times \mathcal{P}(X) \times \mathcal{Q}$ est un ensemble fini de transitions. Une transition de l'automate ; $e = (q, g, a, r, q') \in E$ représente une transition de q vers q', la garde g contient une condition dans $\mathcal{C}(X)$ qui doit être satisfaite afin que la transition soit franchissable, $a \in \Sigma$ est le nom d'une action et r est l'ensemble des horloges devant être remises à zéro. La transition e s'écrit $q \xrightarrow{g,a,r} q'$,
- $\mathsf{Inv} : \mathcal{Q} \to \mathcal{C}(X)$ est une fonction qui associe un invariant $\mathsf{Inv}(q)$ à chaque état de contrôle q. Un invariant ne contient habituellement que des conditions de la forme $x \leq c$ ou $x < c$ qui doivent être satisfaites lorsque le temps s'écoule dans l'état de contrôle considéré.

Les automates temporisés constituent un modèle dont la sémantique s'exprime en termes de systèmes de transitions temporisés [63]. Une configuration (ou état) d'un automate temporisé est un couple (q, v), avec q un état de contrôle, et v une valuation d'horloges. La configuration initiale (q_0, v_0) d'un automate temporisé est telle que $v_0(x) = 0$ pour toute horloge x. À partir d'une configuration (q, v) deux types de transitions sont possibles :

- une transition de durée qui consiste à rester dans le même état de contrôle q et à incrémenter les valeurs des horloges uniformément de d unités de temps, notée $(q, v) \xrightarrow{d} (q, v + d)$. La transition n'est possible que tant que $v + d$ vérifie l'invariant $\mathsf{Inv}(q)$. Les transitions de durée de l'exemple de la figure 3.2 sont représentées par des liaisons en pointillés bouclant sur leur état de contrôle respectif (Souvent, ces transitions sont implicites et ne sont pas représentées). Depuis l'état de contrôle Lumière allumée, l'incrément de la valuation de l'horloge x par franchissement de la transition de durée n'est possible que tant que celle-ci reste inférieure à 100 unités de temps (invariant de l'état de contrôle).
- une transition d'action, notée $(q, v) \xrightarrow{a} (q', v')$, associée à la transition $q \xrightarrow{g,a,r} q'$, si v vérifie la garde g. Si une horloge x de v appartient à r, elle est remise à zéro tel que $v'(x) = 0$. Sinon elle conserve sa valeur, telle que $v'(x) = v(x)$. De plus v' doit vérifier l'invariant $\mathsf{Inv}(q')$. Sur l'exemple de la figure 3.2, la transition d'action depuis l'état de contrôle Lumière éteinte vers l'état de contrôle Lumiére allumée est franchie si la garde *poussoir* est vérifiée. Le franchissement s'accompagne de la remise à zéro de la valuation de l'horloge x.

3.2 Modélisation sous Uppaal

Afin de permettre la lecture des modèles que nous proposons, cette section détaille le minimum de la syntaxe et de la sémantique utilisé. Une description complète de ce que propose Uppaal peut être trouvée dans [62] et [90] .

La variante Uppaal des automates temporisés décrit dans la sous-section 3.1.3 manipule également des variables entières bornées et ajoute une contrainte d'urgence de franchissement d'une transition. L'interface de modélisation couplée à cette variante offre des commodités de modélisation qui permettent d'améliorer la concision et la lisibilité du modèle. Par exemple, la manipulation d'une variable entière évite au modélisateur d'avoir à construire un automate associant un état à chaque valeur possible de la variable et de gérer les transitions.

La figure 3.3 représente les différents éléments qui permettent de modéliser un système sous Uppaal.

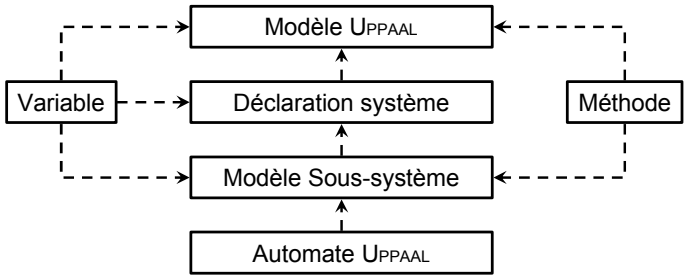


Figure 3.3 – Diagramme d'association d'un modèle Uppaal

Une modélisation sous UPPAAL consiste à décrire des classes de sous-systèmes par l'intermédiaire d'automates UPPAAL manipulant des variables et des méthodes privées. Afin de modéliser l'interaction des sous-systèmes, les automates UPPAAL ont la possibilité de communiquer par l'intermédiaire de variables globales et d'utiliser des méthodes communes. La déclaration proprement dite du modèle consiste finalement à instancier les différents automates UPPAAL déclarés dans un modèle UPPAAL du système étudié.

L'atelier logiciel se charge de procéder à la composition du modèle UPPAAL pour le traduire en un 6-uplet $\langle X, \mathcal{Q}, q_{init}, \Sigma, E, \mathsf{Inv}\rangle$ décrivant un automate temporisé $\mathcal{A}$. Les sous-sections suivantes décrivent les différents éléments de la syntaxe UPPAAL.

3.2.1 Automate Uppaal

Le comportement d'une classe de sous-système peut être modélisé sous la forme d'un automate UPPAAL. Un automate UPPAAL est un automate à états dont la syntaxe a été enrichie pour faciliter la modélisation. Les différentes déclarations possibles d'états sont représentées sur la figure 3.4.

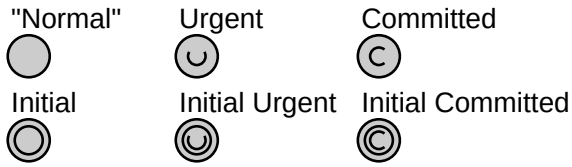


FIGURE 3.4 – Les différentes déclarations d'état utilisables sous UPPAAL

Un état peut être déclaré comme étant :
- Initial. c'est l'état de contrôle initial (et unique) de l'automate UPPAAL,
- Urgent. Lorsqu'un automate UPPAAL est dans un état Urgent, toute évolution du temps est interdite. Par conséquent, toutes les transitions franchies durant l'activité de l'état Urgent sont considérées comme franchies à temps nul.
- Committed. Lorsqu'un des automates UPPAAL d'un modèle est dans un état Committed, toute évolution du temps est interdite comme pour les états Urgent. Mais de plus, les seules transitions autorisées à être franchies sont celles qui permettent à un des automates UPPAAL du modèle de quitter un état actif Committed.

De plus, il est possible d'associer un invariant à chaque état. Celui-ci doit être vérifié tant que l'état est actif. La figure 3.5 représente un exemple pour lequel l'invariant horloge<=100 est associé à l'état S2.

Les transitions entre états offrent également des facilités de modélisation. Outre l'état de départ et l'état d'arrivée qu'elle relie, une transition permet de déclarer :
- une garde (ou condition de franchissement de transition) qui doit être vérifiée pour que la transition soit franchie. Notons que la vérification de la garde n'impose pas le franchissement. C'est pourquoi un automate UPPAAL est dit non déterministe. Dans l'exemple de la figure 3.5, l'état S1 peut rester indéfiniment actif que a soit égal à 1 (garde *a==1*) ou différent de 1 (garde *a!=1* vérifiée).
- une étiquette de synchronisation. Celle-ci permet de synchroniser l'évolution de deux automates UPPAAL différents par l'intermédiaire d'un canal de synchronisation. L'utilisation des canaux de synchronisation est décrite dans la sous-section 3.2.2.

Dans l'exemple de la figure 3.5, la transition de l'état `S3` à l'état `S1` est franchie à la réception de la synchronisation `top[i]`.

– un ensemble d'actions. Les actions d'une transition vont être effectuées au franchissement de cette dernière. Ces actions consistent en la mise à jour de variables de manière explicite ou par l'intermédiaire de l'exécution de méthodes. Dans l'exemple de la figure 3.5, le franchissement de la transition entre l'état `S3` et l'état `S1` s'accompagne d'une mise à jour de la variable b à 2 et de l'exécution de la méthode `methode()`.
– la sélection de la valeur d'une variable dans une liste prédéfinie. Ce champ permet de décrire une transition dont les champs condition, synchronisation et action dépendent de la valeur sélectionnée. Ce champ économise la saisie d'autant de transitions que de valeurs possibles de la variable sélectionnée entre deux états. Dans l'exemple de la figure 3.5, la variable de sélection i peut prendre les valeurs entières 0, 1, 2 ou 3. Par conséquent, la transition de l'état `S3` à l'état `S1` est en fait franchie à la réception de la synchronisation `top[0]`, `top[1]`, `top[2]` ou `top[3]`. Cette syntaxe est spécifique à Uppaal mais elle est facile à traduire sous la forme d'un automate possédant plusieurs transitions possibles entre deux états.

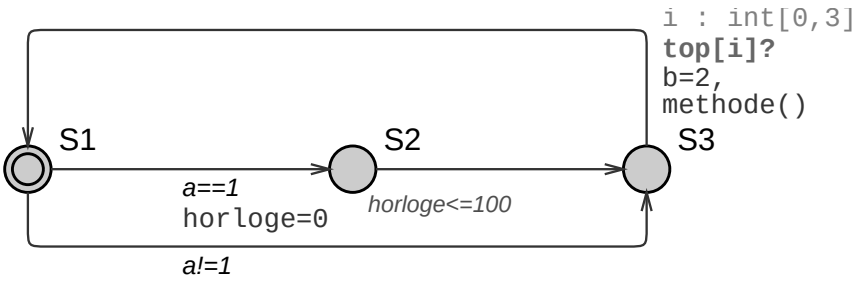


Figure 3.5 – Exemple d'automate Uppaal

En plus des variables d'horloge et des états recensés dans la description formelle d'un automate temporisé, cette description d'un automate Uppaal et ces derniers exemples font référence à d'autres types d'objets tels que les canaux de synchronisation. La section suivante détaille les types d'objet offerts par Uppaal et utilisés dans la modélisation de la solution EPL-HA.

3.2.2 Variables et méthodes

Nous avons évoqué la manipulation de variables et méthodes privées et globales par les automates Uppaal. Uppaal permet de déclarer différents types d'objets, comme les horloges ou les entiers aperçus dans l'exemple de la figure 3.5. La traduction du modèle Uppaal en automate temporisé se chargera d'interpréter les combinaisons de valeurs prises par ces différents objets en états de contrôle d'un automate temporisé.

Les méthodes vont nous permettre de décrire des actions de mise à jour évoluées au franchissement d'une transition, mais aussi de regrouper des mises à jour simples pour alléger l'aspect des automates Uppaal. La déclaration des objets, des méthodes et les expressions utilisent une syntaxe inspirée des langages C, C++ ou JAVA. Par contre, seule une partie des expressions offertes par ces langages sont implémentées dans Uppaal.

Notre modélisation utilise différents types d'objets gérés par Uppaal dans les invariants d'états, les différents champs d'une transition et les méthodes.

Horloges (`clock`). Les variables d'horloge UPPAAL peuvent être déclarées comme variables globales du modèle UPPAAL ou comme variables privées d'un automate UPPAAL (Par exemple `clock TcycClock ;`).

Entiers bornés (`int`). La déclaration de la variable entière `LateMediumCounter` s'écrit `int LateMediumCounter ;`. Il est également possible de donner un domaine de valeur de la variable ainsi qu'une valeur initiale. La déclaration `int [1,2] FirstReceivingPort=1 ;` crée la variable entière `FirstReceivingPort` dont les valeurs possibles sont 1 et 2 et dont la valeur à l'initialisation de l'automate est 1. Il est également possible de créer une variable constante, par exemple la variable entière `AllowedDelay` de valeur constante 50 avec l'expression `const int AllowedDelay=50 ;`.

Canaux de synchronisation binaire (`chan`). Les canaux de synchronisation ne sont utilisables que dans les étiquettes de synchronisation des transitions. Ils sont déclarés en tant qu'objets globaux dans le champ de déclaration du système et permettent à deux automates UPPAAL instanciés de coordonner le franchissement d'une de leurs transitions. On considère deux automates UPPAAL utilisant le canal de synchronisation `DatagramSync` (déclaré par `chan DatagramSync ;`). L'instance de l'automate UPPAAL dont la transition est étiquetée `DatagramSync!` est dite émettrice de la synchronisation. Elle ne peut se synchroniser qu'avec un unique récepteur dont la transition est étiquetée `DatagramSync?`. Si la garde respective de ces deux transitions est vérifiée, les deux transitions sont franchies en même temps ; les deux automates se synchronisent. Si plusieurs couples d'automates d'un même modèle peuvent se synchroniser depuis une configuration, l'ordre des synchronisations est réalisé de manière non déterministe. L'évolution du modèle peut être bloquée si un automate envoie un signal `DatagramSync!`, mais que son récepteur ne peut pas le recevoir parce que sa garde ou son état d'activation rendent la transition non franchissable.

Canaux de synchronisation multiple (`broadcast chan`). Les canaux de synchronisation multiple (ou canaux de diffusion) associent un émetteur et plusieurs récepteurs. Pour un canal de synchronisation multiple `FrameSync` (déclaré par `broadcast chan FrameSync ;`), les récepteurs dont la transition est étiquetée `FrameSync?` et est franchissable vont se synchroniser à l'envoi du signal `FrameSync!` par l'émetteur. Par contre, les canaux de synchronisation multiple ne peuvent pas bloquer l'évolution d'un modèle UPPAAL. L'émetteur n'a pas besoin que ses récepteurs puissent se synchroniser pour franchir sa transition étiquetée `FrameSync!`.

Tableaux. Des tableaux d'horloges, de booléens, d'entiers, de binaires, de canaux de synchronisations binaires et multiples peuvent être déclarés afin d'organiser la description d'un modèle. Par exemple, l'expression `bool LinksBetweenIDS [3,4]` crée une matrice de booléen composée de 3 lignes et 4 colonnes. Par contre, la valeur du booléen situé à la 1ère ligne et à la 3e colonne est accessible avec `LinksBetweenIDS[0][2]`.

Types personnels. Des types personnels d'objet dérivés des types précédents peuvent être déclarés afin d'organiser la description d'un modèle. L'expression `typedef int [0,10] idNode ;` déclare le type d'objet `idNode` dérivé du type entier tel que les valeurs possibles sont comprises entre 0 et 10.

Structures. Des structures regroupant différents types d'objet précédents peuvent être déclarées afin d'organiser la description d'un modèle. L'expression `typedef struct idMsg Msg ;idNode Addr ; FrameDefinition ;` définit la structure `FrameDefinition` composée d'un objet de type `idMsg` et d'un objet de type`idNode`. Les valeurs de l'objet `Trame` déclaré avec l'expression `FrameDefinition Trame ;` sont accessibles avec `Trame.Msg` et `Trame.Addr`.

3.2.3 Déclaration du système

À partir des différents éléments de modélisation décrits jusqu'à présent, nous avons souligné comment il est possible de décrire un modéle UPPAAL où différents comportements interagissent en créant un automate par comportement. Mais dans le cas où le même sous-système peut être utilisé (à des valeurs de constantes près) pour différents acteurs du modèle, la déclaration d'un système UPPAAL permet également de décrire un modèle par instanciation de sous-systèmes UPPAAL.

Considérons une traduction du modèle du sous-système d'éclairage de la figure 3.2 sous UPPAAL. L'automate UPPAAL représenté en figure 3.6 est celui du sous-système `eclairage(int identifiant)`, fonction de la variable entière `identifiant` et utilisant l'objet privé `clock x` et l'objet global `chan poussoir[3]`. On désire obtenir un modèle constitué de trois sous-systèmes d'éclairages et d'un sous-système d'envoi des commandes poussoir respectives. Ce dernier est modélisé par un sous-système `Commande` (non représenté) capable d'émettre dans les canaux de synchronisation binaire poussoir(0), poussoir(1), poussoir(2).

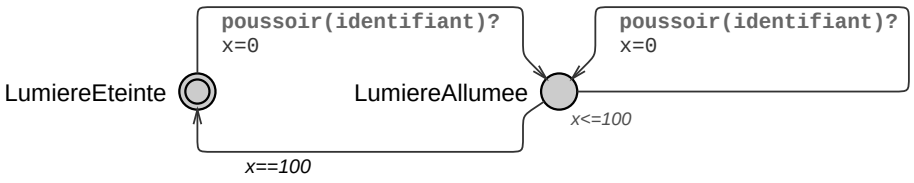


FIGURE 3.6 – Automate UPPAAL du sous-système eclairage

Pour générer trois instances du sous-système UPPAAL `eclairage`, il suffit d'écrire l'expression `Illumination (int [0,2] id) = eclairage(identifiant) ;` dans le champ de déclaration du système.

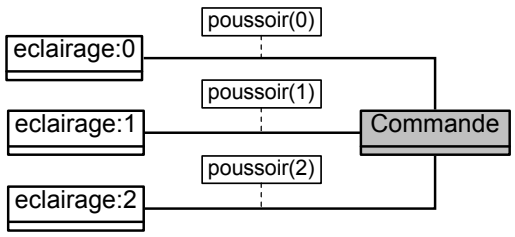


FIGURE 3.7 – Diagramme d'objets du système `Illumination` modélisé

Finalement, un système UPPAAL est déclaré avec l'expression `system` suivie de la liste des sous-systèmes le constituant. L'expression `system Illumination, Commande ;` déclare un système UPPAAL dont le diagramme d'objets est illustré par la figure 3.7.

Le système est constitué des sous-systèmes UPPAAL `eclairage(0)`, `eclairage(1)` et `eclairage(2)` précédemment créés et du sous-système `Commande`.

3.2.4 Ecriture des propriétés à vérifier

Les algorithmes de model-checking, tels que ceux utilisés par l'exécutable `verifyta` de l'atelier logiciel UPPAAL, permettent de vérifier des propriétés sur le comportement du système modélisé [64]. Ces propriétés sont exprimées par l'intermédiaire d'une logique temporelle qui permet d'écrire de manière formelle des expressions du type *Il existe un état tel que...*, *Il y a toujours...* ou *...implique...*. Suivant la logique temporelle utilisée, il est possible de vérifier différents types de propriété :

- l'atteignabilité : Il est possible d'atteindre une certaine situation ;
- la sûreté : Pour des conditions données, une situation ne sera jamais atteinte ;
- la vivacité : Pour des conditions données, une situation finira par être atteinte ;
- l'équité : Pour des conditions données, une situation aura lieu un nombre infini de fois.

Les exigences fonctionnelles énoncées dans la section 2.4 expriment des situations que l'on souhaite ne jamais rencontrer. Par conséquent, elles font appel à des propriétés de sûreté du système que l'on souhaite prouver.

Nous utilisons la logique temporelle adaptée à l'algorithme de model-checking utilisé pour traduire les exigences fonctionnelles énoncées dans la section 2.4 en propriétés de sûreté. UPPAAL permet d'utiliser un sous-ensemble de la logique temporelle Timed Computation Tree Logic (TCTL) proposée par Alur, Courcoubetis et Dill [65]. TCTL est une extension de la logique temporelle CTL [66] [67] qui suffit à notre utilisation.

Soient p et c des évaluations d'expressions booléennes portant sur les valeurs de variables ou l'activité des états d'un modèle :

- la propriété $E<>p$ est vérifiée pour un modéle si et seulement il existe un séquence d'évolution du modèle menant à un état pour lequel p est vraie.
  $E<>$ permet d'exprimer des propriétés d'atteignabilité.
- la propriété $A[]p$ est vérifiée si et seulement si p est vraie pour tous les états atteignables du modèle. Elle traduit l'invariance. L'invariance permet d'exprimer des propriétés de sûreté sous la forme $A[]$ *not* c (vérifiée si c est fausse pour tous les états atteignables du système). Nous utilisons uniquement cette forme pour modéliser nos exigences fonctionnelles.
- la propriété $E[]p$ est vérifiée pour un modèle si et seulement si :
  - il existe une séquence d'évolution du modèle pour lequel p est toujours vrai et le nombre d'états est infini,
  - ou l'état final du modèle est tel que son invariant et p sont satisfait pour toute transition de durée,
  - ou il n'y a pas de transition possible.

  $E[]$ permet d'exprimer des propriétés d'équité.
- la propriété $A<>p$ est vérifiée si et seulement si toutes les séquences d'évolution permettent d'atteindre un état pour lequel p est vraie. La propriété $p-->q$ est vérifiée si toute séquence pour laquelle p devient vraie peut conduire ensuite à avoir q vraie. $A<>$ et $-->$ permettent d'exprimer des propriétés de vivacité.

Il n'est pas toujours possible d'énoncer les évaluations booléennes en fonction de variables du modèle, surtout lorsque cette expression dépend de l'historique d'une situa-

tion. Il arrive donc que l'on ait recourt à des automates observateurs, synchronisés avec l'évolution du modèle de système. Les expressions booléennes évaluées utilisent alors des variables des automates observateurs qui témoignent de la situation ou de l'historique des situations du modèle.

Les automates observateurs sont également utilisés pour restreindre le comportement d'un modèle et limiter le nombre d'états atteignable. Nous n'utilisons pas cette méthode d'abstraction dans notre modèle.

3.3 Modéle générique d'une architecture sous Uppaal

Le réseau que nous étudions utilise un nombre fini de références de composants. Cette caractéristique se prête bien à la démarche de modélisation sous UPPAAL décrite dans la section précédente. Elle nous a également permis d'inclure de la modularité au modèle.

Le modèle UPPAAL d'une architecture EPL-HA sera décrit par instanciation paramétrée dans une liste finie de sous-systèmes UPPAAL. Chaque sous-système UPPAAL de la liste utilise un automate UPPAAL pour modéliser le comportement d'un ou plusieurs acteurs de l'architecture. Dans la suite du document, sous-système UPPAAL et automate UPPAAL seront confondus. Le diagramme des classes de la figure 3.8 suivante illustre la structure que nous avons mise en place sous UPPAAL pour la modélisation d'architectures EPL-HA.

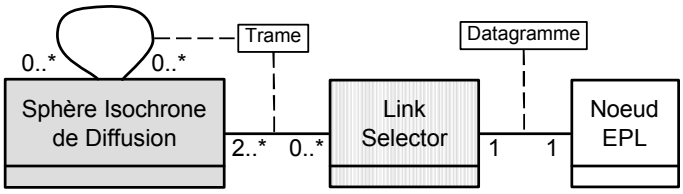


FIGURE 3.8 – Diagramme des classes UML de la modélisation

Nous avons retenu trois classes de comportement qui vont être détaillées dans les sous sections suivantes :

- la classe nœuds EPL, qui modélise le comportement au niveau 2 du modèle OSI (cf. 2.1) défini par l'EPSG dans [76] et [86],
- la classe LS, qui modélise la sélection et la duplication de trame (cf. 2.3) définies par la spécification [86],
- EPL utilise de préférence un réseau partagé. La modélisation est restreinte exclusivement à ce type de réseau (n'utilisant pas de commutateurs, mais des composants de diffusion tels que des concentrateurs et des convertisseurs Fibre-Cuivre). Par conséquent, tous les composants d'un médium appartiendront au même domaine de diffusion. Notre approche consiste à découper un domaine de diffusion en une ou plusieurs parties que nous appelons Sphère Isochrone de Diffusion que nous modélisons par la classe Isochronous Diffusion Sphere (IDS). La classe IDS modélise l'introduction d'une latence de transmission des trames par les équipements d'interconnexion compris dans l'IDS. Cette latence peut être assimilée à la valeur du rayon de la sphère.
  En effet, chaque équipement d'interconnexion, y compris un câble, nécessite du temps pour transmettre un bit reçu sur une de ses prises à une autre ou plusieurs autres de ses prises. Ce temps est dû à la vitesse de propagation du signal électrique dans les conducteurs ou au temps de traitement de ce signal.

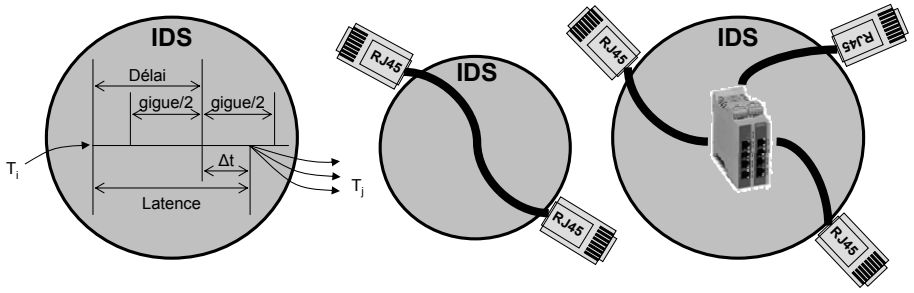


FIGURE 3.9 – Illustration de la fonction IDS et deux exemples de regroupement en IDS

Comme illustré dans la partie gauche de la figure 3.9, la latence entre une trame entrante T_i sur un des ports et sa réplication T_j sur les autres ports est égale à $Delai + \triangle t$ avec $|\triangle t| \leqslant \frac{gigue}{2}$. Les valeurs de délai et de gigue d'une IDS seront déduites des valeurs respectives des équipements regroupés.
La classe IDS doit nous permettre d'abstraire la modélisation d'un médium en regroupant plusieurs équipements d'interconnexion au sein d'une seule instance. La figure 3.9 donne deux exemples de regroupements, une IDS ne contenant qu'un câble et une IDS regroupant un concentrateur et du câble. La modélisation indépendante et précise des équipements d'interconnexion conduirait à un grand nombre de combinaisons des états respectifs de chacun. Notre objectif est de limiter l'explosion combinatoire qui en résulterait. En effet, cette précision ne sera pas exploitée lors de la modélisation des propriétés attendues.

Les associations entre les différentes classes de composants modélisent les échanges réalisés entre ceux-ci qui sont susceptibles de faire évoluer leur comportement. Dans la réalité, ces échanges correspondent à des échanges de données. Le haut de la figure 3.10 donne un exemple d'architecture avec deux nœuds connectés par l'intermédiaire d'un concentrateur. Ces deux nœuds possèdent la fonction LS intégrée bien que le médium ne soit pas redondé. Sur le bas de la figure, nous avons représenté les échanges sur cet exemple du point de vue du modèle OSI. Pour nommer les différents types d'échange, la terminologie que nous utilisons diffère de celle définie par le modèle OSI.

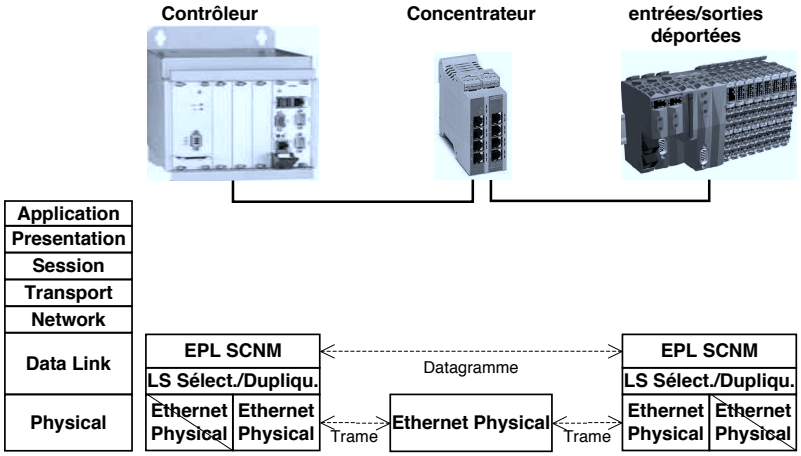


FIGURE 3.10 – Echanges du point de vue du modèle OSI

– Nous appelons datagrammes les échanges entre différentes couches SCNM d'un réseau. La couche SCNM gère le comportment d'un nœud EPL. Par conséquent, un datagramme correspond à l'entité transmise entre un nœud EPL et son LS. L'échange de datagramme est modélisé par un canal de synchronisation entre l'instance de la classe nœud et l'instance de la classe LS associée.
– Nous appelons trames les échanges entre différentes couches physiques d'un réseau. Par conséquent, un datagramme correspond à l'entité transmise entre LS et composants d'interconnexion du réseau. L'échange de trame est modélisé par un canal de synchronisation multiple entre instances de la classe LS et instances de la classe IDS.

Un canal de synchronisation n'étant porteur d'aucune d'information, une synchronisation de type datagramme ou trame s'accompagne d'une recopie d'informations liées à la synchronisation (par exemple l'identifiant du nœud source ou du(des) nœud(s) destinataire(s)). Dans la suite du chapitre, les expressions de réception ou d'émission d'un datagramme (ou d'une trame) correspondront à la synchronisation sur émission ou réception d'un événement de type datagramme (ou de type trame).

La sous-section suivante détaille l'instanciation du modèle générique à partir d'un exemple d'architecture.

3.3.1 Exemple d'instanciation d'une architecture

À partir des trois classes de comportement énumérées en introduction, il est possible de décrire n'importe quelle architecture utilisant EPL-HA.

Illustrons l'utilisation de ces trois classes de comportement avec l'exemple d'architecture représenté par la figure 3.11. Cette architecture est constituée de trois nœuds EPL-HA communiquant par l'intermédiaire d'un médium redondant.

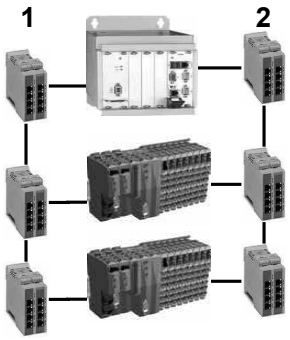


FIGURE 3.11 – Exemple d'architecture.

Comme le montre le diagramme de classe 3.8, une instance de nœud communiquera par l'intermédiaire d'un datagramme avec un et un seul LS. Pour un nœud déclaré, le modèle générique créé une instance de la classe nœud ainsi que l'instance de la classe LS et le datagramme de communication correspondant, indépendamment de la topologie de l'architecture. En ce qui concerne l'exemple de la figure 3.11, nous obtenons trois groupes d'objets. L'identifiant, unique pour un groupe d'objets et compris entre 0 et 2, est choisi par l'utilisateur. La figure 3.12 suivante illustre le résultat de la déclaration des trois nœuds.

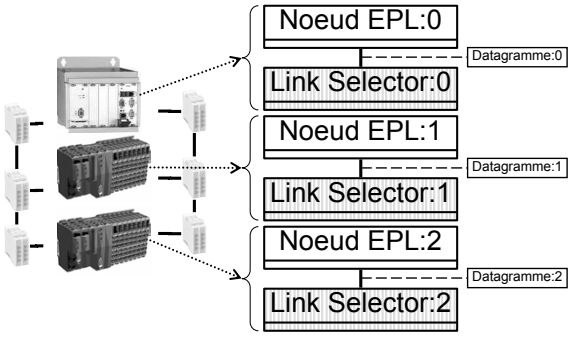


FIGURE 3.12 – Instanciation des modèles de nœud et LS pour l'architecture de la figure 3.11.

Chaque instance d'IDS possède un identifiant unique et différent des identifiants de nœuds.

Afin de limiter la taille du modèle à vérifier, on pourrait abstraire chaque domaine de diffusion de l'exemple de la figure 3.11 en une seule IDS qui introduirait le même délai entre un nœud émetteur et les 2 nœuds potentiellement récepteurs. Le délai est effectivement le même entre le nœud 1 et les nœuds 0 ou 2. Il correspond au délai pour que le signal parcoure 2 concentrateurs et une longueur de câble similaire. Par contre, le signal ne suivra pas des chemins similaires, lorsque 0 est émetteur, pour aller jusque 1 ou jusque 2. L'abstraction d'un domaine de diffusion à une IDS ne serait donc pas appropriée.

Autre limite à l'abstraction, le choix d'un découpage de domaines de diffusion en IDS nécessite que le délai maximal introduit par une IDS soit inférieur au délai de réaction minimal des nœuds. Une IDS ne peut donc pas englober trop d'équipements d'interconnexion sur une topologie étendue.

A l'inverse, un découpage possible de l'exemple donné par la figure 3.11 est le partage des deux domaines de diffusion en autant d'IDS que de composants d'interconnexion (Concentrateurs et câbles). Chaque médium serait alors modélisé par 8 IDS. Néanmoins, si un domaine de diffusion est découpé en un nombre d'IDS trop important, le modèle de l'architecture risque d'être trop gros pour être capable de vérifier des propriétés.

La difficulté pour la personne utilisant le modèle générique sera donc essentiellement de trouver le bon compromis sur le découpage d'un domaine de diffusion en IDS. Nous choisissons ici de découper les deux domaines de diffusion en autant d'IDS que de concentrateurs. Chaque IDS modélise par conséquent un concentrateur et une partie des câbles reliant le concentrateur à ses voisins. La figure 3.13 illustre le résultat du découpage.

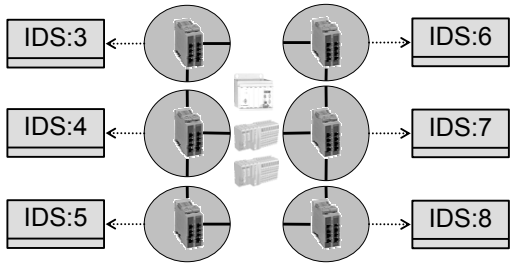


FIGURE 3.13 – Instanciation des IDS pour l'architecture de la figure 3.11.

Comme le représente le diagramme des classes 3.8, une instance de la classe IDS échange des trames avec d'autres instances de la classe IDS et des instances de la classe LS. À ce stade de la déclaration de l'architecture, nous avons d'une part des couples regroupant un nœud et son LS et d'autre part des IDS. Ceux-ci ne peuvent pas communiquer tant que les associations de type trame n'ont pas été déclarées.

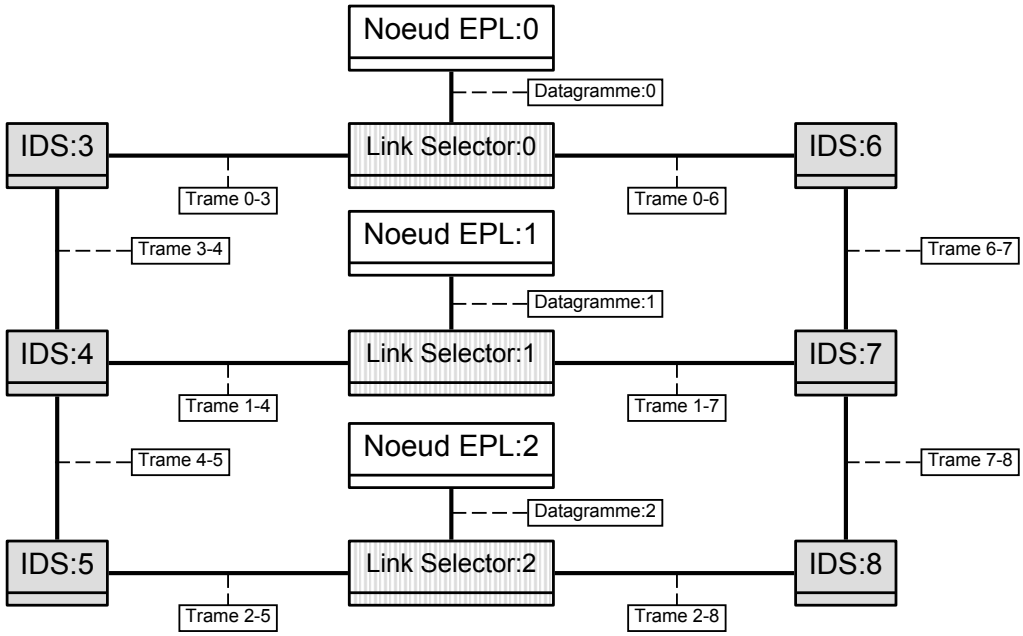


FIGURE 3.14 – Diagramme d'objet complet pour l'architecture de la figure 3.11.

Ces associations décrivent les connexions physiques qui relieraient dans la réalité les composants ou les ensembles de composants modélisés et leur permettraient d'échanger des trames ou des datagrammes entre couches connectées. La figure 3.14 donne le diagramme d'objet complet après complément de ces connexions manquantes. Nous traduisons ces connexions sous la forme de deux matrices symétriques.

La première matrice décrit les connexions physiques de chaque LS avec chaque IDS et indique à quel médium ces IDS correspondent. Le tableau suivant donne la composition de la matrice correspondant au diagramme de la figure 3.14

NodePortsAssignemnt	IDS 3	IDS 4	IDS 5	IDS 6	IDS 7	IDS 8
LS 0	1	0	0	2	0	0
LS 1	0	1	0	0	2	0
LS 2	0	0	1	0	0	2

TABLE 3.1 – Valeurs de la matrice décrivant les connexions entre les LS et les IDS

La valeur 0 indique que l'échange direct de trames entre le LS et l'IDS n'est pas autorisé. La valeur 1 indique que l'échange entre le nœud et l'IDS est autorisé et correspond au port 1 du nœud. La valeur 2 indique que l'échange entre le nœud et l'IDS est autorisé et correspond au port 2 du nœud.

La seconde matrice décrit les connexions physiques entre IDS et indique à quel médium

ces IDS correspondent. Le tableau suivant donne la composition de la matrice correspondant à la figure 3.14

LinksBetweenIDS	IDS 3	IDS 4	IDS 5	IDS 6	IDS 7	IDS 8
IDS 3	0	1	0	0	0	0
IDS 4	1	0	1	0	0	0
IDS 5	0	1	0	0	0	0
IDS 6	0	0	0	0	2	0
IDS 7	0	0	0	2	0	2
IDS 8	0	0	0	0	2	0

TABLE 3.2 – Valeurs de la matrice décrivant les connexions entre les IDS

La valeur 0 indique que l'échange direct de trame entre les deux IDS n'est pas autorisé. La valeur 1 indique que l'échange entre les deux IDS est autorisé et qu'elles appartiennent au médium 1. La valeur 2 indique que l'échange entre les deux IDS est autorisé et qu'elles appartiennent au médium 2.

Ces deux matrices font partie de l'ensemble de paramètres servant à décrire une architecture et utilisés comme entrées de la modélisation générique. Ces paramètres servent d'une part à décrire l'architecture spécifique que l'on souhaite vérifier et d'autre part à indiquer quel comportement de défaillance est autorisé. Tous les paramètres sont réunis au début du fichier de la modélisation (un exemple du code associé à ces paramètres est donné en annexe 4.4). Aucune modification sur les automates UPPAAL ou la déclaration du système n'est nécessaire. Les automates et la déclaration du système (à partir des paramètres) sont génériques. Les trois sous-sections suivantes détaillent les classes que nous avons retenues, leurs rôles, ainsi que les paramètres associés à chacune d'elles.

3.3.2 Classe Isochronous Diffusion Sphere

La fonction principale des composants d'interconnexion qui nous intéresse est l'introduction de délais dans la transmission d'une trame. Nous voulons également simuler une défaillance se traduisant par la non-transmission de la trame.

Le diagramme de séquence de la figure 3.15 donne le scénario recherché pour la fonction introduction de délai en s'appuyant sur l'architecture de la figure 3.11 :

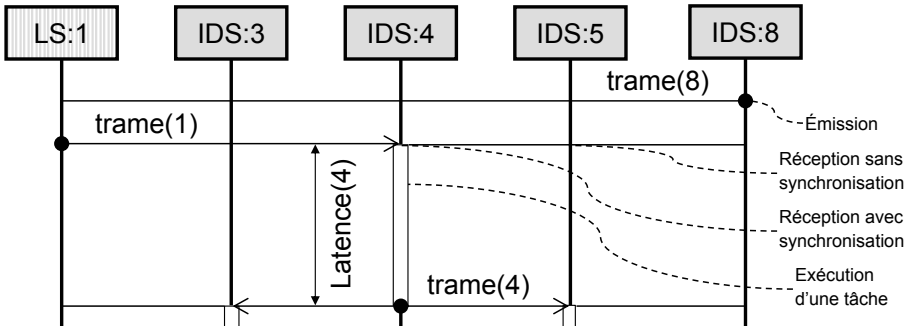


FIGURE 3.15 – Exemple de séquence transmission de trame entre IDS et LS.

Si une trame est émise par une instance de la classe IDS ou de la classe LS mais ne correspond pas à un lien physique dans la réalité, un filtrage prévient la réception de l'instance. Dans l'exemple, IDS:8 initie une trame (trame(8)), mais IDS:3, IDS:4, IDS:5 et LS:1 ne synchronisent pas, car le filtrage les en empêche.

Si une trame est émise par une instance de la classe IDS ou de la classe LS et correspond à un lien physique dans la réalité, le filtre autorise la réception par l'instance et retarde la transmission d'une valeur de latence variable.

La latence est déterminée à partir des paramètres de délai et de gigue (tels que définis dans la figure 3.9). À l'issue de la latence, une trame conservant les mêmes informations est émise en veillant à ne pas générer d'écho (sans renvoyer l'événement vers sa source). Dans l'exemple, LS:1 initie une trame (trame(1)) avec laquelle IDS:4 synchronise en déclenchant une tâche de retardement de la transmission. A l'issue de cette tâche, la trame de transmission trame(4) est initiée.

À l'issue de ce scénario, nous pouvons mettre en évidence une partie des paramètres qui servent à instancier la classe IDS :

- un identifiant unique qui permet un filtrage par les autres instances,
- une règle de filtrage des autres instances correspondant à la matrice 3.2 de l'exemple,
- une valeur moyenne de latence pour définir le temps d'attente,
- une valeur de gigue pour définir le temps d'attente.

Les deux derniers paramètres doivent être déterminés en fonction des composants réseaux englobés dans l'instance d'IDS considérée. Pour le découpage de la figure 3.13, les valeurs de latence moyenne et de gigue sont égales respectivement à la latence moyenne et à la gigue introduites par un concentrateur et une longueur faible de câble.

La modélisation de la défaillance de l'instance consiste à empêcher la réception de toute trame. Nous avons également modélisé la possibilité de réparation après défaillance pour assurer de nouveau le transfert de trames. Nous voulons vérifier de cette manière que, par exemple, le sectionnement d'un câble comme son remplacement, ne peut perturber le cycle des échanges EPL ou la redondance logicielle. Par conséquent, une instance d'IDS utilise deux paramètres supplémentaires :

- une autorisation de défaillance,
- une autorisation de réparation (qui n'a de sens que s'il y a défaillance)

3.3.3 Classe Link Selector

Les fonctions remplies par un LS qui nous intéressent sont la duplication de trame sortante et la sélection de trames entrantes. La combinaison de ces deux fonctions doit permettre d'assurer la redondance matérielle spécifiée par l'extension EPL-HA (section 2.3).

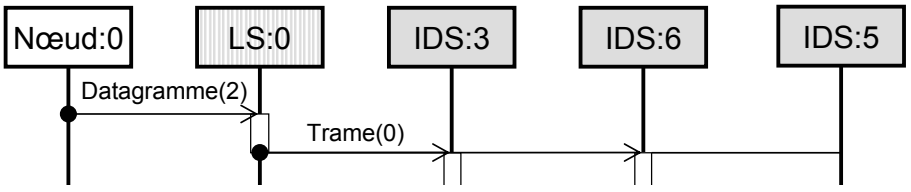


FIGURE 3.16 – Exemple de séquence duplication de trame dans un LS.

À la réception d'un datagramme venant du nœud, la fonction duplication de trame initie une trame (cf. diagramme de séquence de la figure 3.16). Le diagramme de séquence de la figure 3.17 applique un scénario de sélection à l'exemple de la sous-section 3.3.1.

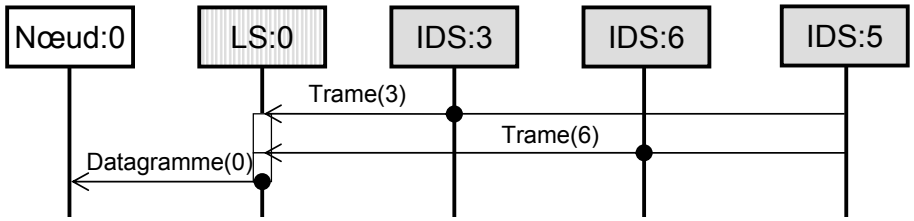


FIGURE 3.17 – Exemple de séquence sélection de trame dans un LS.

À la première trame reçue, non soumise à blocage par un filtre similaire à celui décrit pour la classe IDS, une instance de classe LS va attendre la réception de la trame redondante sensée venir du médium redondant. Afin d'éviter tout blocage, un délai maximal limite l'attente de la trame redondante (en cas de rupture de câble par exemple). À la réception de la trame redondante (cas de la figure 3.17), ou à l'échéance du délai maximal, un datagramme vers le nœud est émis.

À l'issue de ces scénarios, nous pouvons mettre en évidence une partie des paramètres qui servent à instancier la classe LS :
- un identifiant unique qui permet un filtrage par les autres instances,
- une règle de filtrage des autres instances correspondant à la matrice 3.1 de l'exemple,
- une valeur de délai maximal d'attente de la trame redondante.

Nous avons choisi de fixer une valeur constante commune à toutes les instances de LS. En effet dans le cas de LS non intégrés au nœud et non configurables par le réseau, nous ne désirons pas retenir l'hypothèse que des LS configurés hors ligne avec des valeurs optimisées selon leur position seront obligatoirement installés aux endroits et sur les nœuds prévus.

La modélisation ne prend pas en compte la défaillance d'un LS. Celle-ci se traduirait par la non-transmission des trames et des datagrammes reçus et l'isolement du nœud desservi. Du point de vue des autres nœuds, cela correspondrait à la défaillance du nœud. Du point de vue du nœud, l'envoi de datagramme serait impossible et bloquerait toute évolution.

3.3.4 Classe nœud EPL

La classe nœud doit modéliser l'animation des échanges en tant que AMN et la participation aux échanges en tant que SMN. La classe doit également modéliser le passage du comportement SMN au comportement AMN conformément à la redondance logicielle décrite dans la partie 2.2. Enfin, pour nous permettre de vérifier le passage entre ces deux comportements, la classe nœud doit modéliser un état défaillant pour lequel aucun échange n'est assuré.

L'instanciation d'un nœud nécessite par conséquent différents types de paramètres :
- des paramètres de description tels qu'un identifiant unique, mais aussi des données décrivant la réactivité du nœud,
- des paramètres de configuration répertoriés par la spécification [76] et qui indiquent comment est planifié le cycle EPL,

– des paramètres de défaillance pour que l'instance génère des situations nous permettant de vérifier les propriétés liées à la disponibilité.

3.3.5 Modélisation des trames

Nous avons déjà souligné le fait que les échanges de datagrammes ou de trames sont modélisés par des canaux de synchronisation. Or le canal de synchronisation n'est porteur d'aucune autre information. Afin de permettre aux différents comportements de se situer dans le cycle EPL, chaque synchronisation de type trame s'accompagne de la recopie d'une structure d'informations stockée par l'émetteur. Ces informations sont le type de trame et l'identifiant du nœud émetteur initial ou le(s) destinataire(s). Les données de pilotage du processus ne sont pas prises en compte, car elles n'influencent pas le protocole.

La figure 3.18 illustre les premiers échanges d'un cycle EPL en mettant en évidence les différences de perception du cycle du point de vue temporel. Les délais de propagation du signal (amplifiés pour la figure) entre les nœuds, par exemple, conduisent ceux-ci à avoir des dates de début de cycle différentes. Leur position respective dans l'architecture les amène également à percevoir des délais entre trames différents. Dans l'exemple de la figure 3.18 le délai entre PReq CN1 et PRes CN1 est différent entre le MN et les CN.

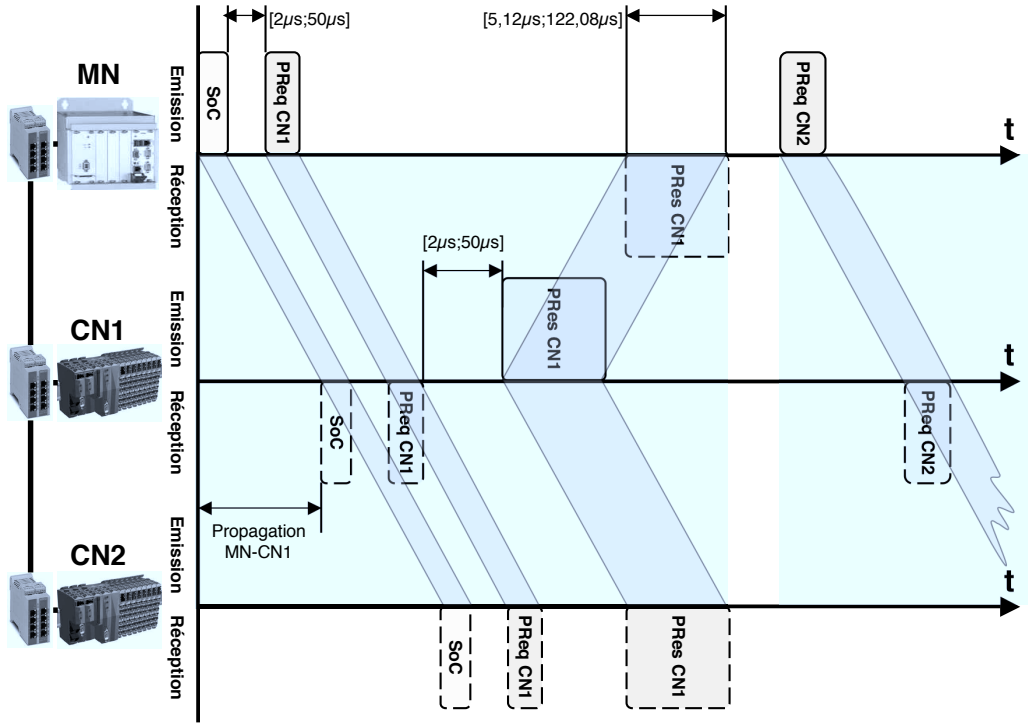


FIGURE 3.18 – Influence des délais de propagation sur la perception du cycle EPL.

Dans la sous-section 2.1.2, nous avons fait remarquer que les délais introduits par la propagation des trames dans l'architecture ainsi que les délais introduits par les données de chacune d'elles étaient significatifs pour représenter le cycle EPL. Ainsi, plus une trame embarque de données applicatives, plus le délai qu'elle introduit est important et plus le temps de cycle nécessaire pour réaliser l'ensemble des échanges augmente. Si la modélisation prend bien en compte ce qui relève de la durée de propagation dans les IDS,

comme nous l'avons précisé dans la sous-section 3.3.2, le délai introduit par une trame est replié (T_{Trame}=0 $\forall$ Trame).

Cette abstraction s'appuie sur les constatations suivantes :

- Sur la figure 3.18, on peut voir que le délai introduit par une trame, c'est-à-dire le délai entre le premier bit et le dernier bit d'une trame est le même, que celle-ci soit émise ou reçue. La planification des échanges d'EPL permet le repli du délai introduit par une trame parallèlement sur tous les nœuds.
- La modélisation considérant uniquement des domaines de diffusion, le délai de propagation d'une trame au travers des composants d'interconnexion du réseau n'est pas fonction de sa taille. Ce serait également le cas si des architectures à base de commutateurs *cut-through* étaient considérées (cf. sous-section 2.1.1). Le repli du délai introduit par une trame n'a donc pas d'influence sur les autres délais constituant le cycle EPL minimum.
- Les échéances (*timeout*) surveillées par un nœud sont chronométrées à partir de la fin de l'émission ou de la réception d'une trame. Elles peuvent être initialisées au début de l'émission ou de la réception d'une trame suivante. Il n'y a de changement d'état d'un nœud au milieu de la réception ou de l'envoi d'une trame. Le repli du délai introduit par une trame nécessite donc d'être pris en compte lors de la configuration des valeurs d'échéances temporelles.

  Pour la modélisation, ces valeurs seront moindres. Par conséquent, la modélisation risque d'augmenter l'exigence de réactivité sur les nœuds mais les conclusions sur les propriétés vérifiées resteront valides.

Le but est de limiter l'explosion combinatoire durant la phase de vérification par abstraction des états et des horloges qui serviraient à gérer les événements de début et de fin de trame tout en préservant la validité de la vérification. La figure 3.19 suivante illustre la conséquence de la non-prise en compte du délai de la trame sur la représentation du cycle EPL et sur sa modélisation. Contrairement à la figure 3.18, les ordres de grandeurs des différents délais sont respectés.

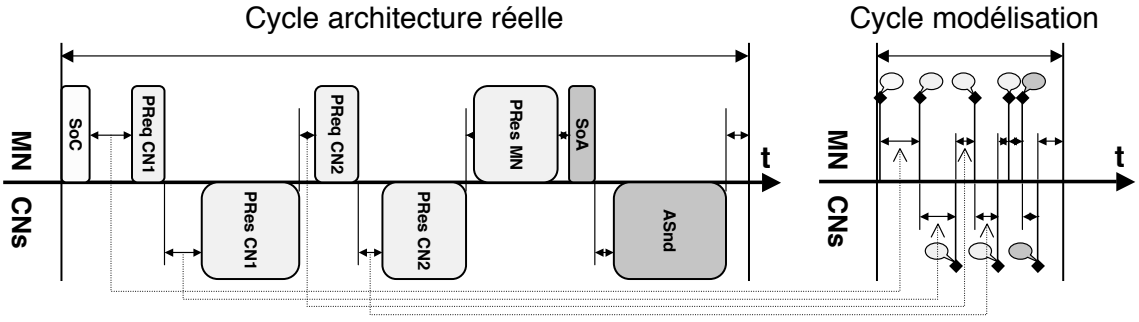


FIGURE 3.19 – Modélisation des trames. Conséquence sur le cycle.

Entre la représentation du cycle sur une architecture réelle et le cycle obtenu par l'intermédiaire de la modélisation, tous les délais de trames sont repliés à zéro. Le délai entre l'émission de deux trames n'est fonction que des délais de propagation du signal et des délais de réaction des nœuds. Par conséquent, le temps de cycle configuré dans le modèle correspondra à un temps de cycle réel réduit, c'est à dire ne prenant pas en compte les durées constantes associées aux trames (à l'instar des échéances temporelles configurées dans dans les nœuds).

3.4 Modèles Uppaal des différents sous-systèmes

3.4.1 Modèle d'un nœud

Description informelle

L'automate UPPAAL d'un nœud EPL, représenté en figure 3.21, modélise la participation aux échanges cycliques spécifiés par le protocole (le même modèle reproduit à une échelle plus grande est donné en annexe 4.4). En fonction de son état actif et des signaux reçus, l'automate UPPAAL du nœud va déduire l'étape du cycle EPL (illustré par la figure 2.2) atteinte et va évoluer en accord avec ce cycle.

Étant donné que nous cherchons à vérifier les conséquences de la défaillance d'un nœud sur la disponibilité des communications, l'automate UPPAAL modélise également la défaillance du nœud. La défaillance qui nous intéresse est l'absence momentanée ou permanente de participation aux échanges. Nous n'avons pas modélisé la défaillance d'un nœud le rendant trop bavard sur le réseau de manière incontrôlable.

La phase de démarrage au cours de laquelle chaque nœud est identifié et est configuré avant de se voir autoriser l'envoi de données applicatives n'est pas considérée par la modélisation. L'état initial modélisé considère que le nœud est immédiatement prêt à participer aux échanges (ou à tomber en panne).

L'automate UPPAAL peut être découpé en trois zones représentant les trois comportements possibles du nœud (cf. 3.20) :

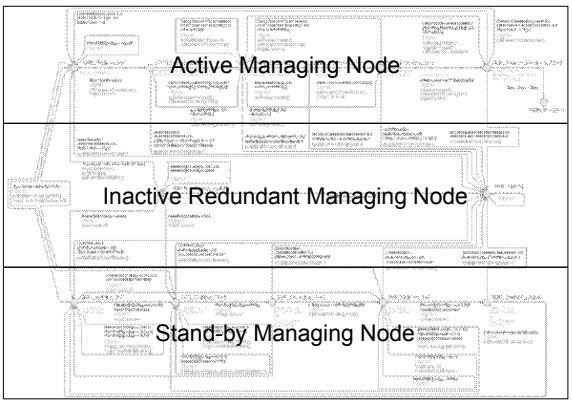


FIGURE 3.20 – Les trois types de comportement d'un nœud.

- animant les échanges en tant que AMN dans les états AMN_*. La succession de ces états respecte le cycle EPL du point de vue AMN illustré dans la figure 2.2. Le nœud annonce le début du cycle avec un SoC avant de demander la production de chacun des autres nœuds à l'aide d'un PReq. À l'issue de ces consultations, le nœud a la possibilité de produire un PRes. À la fin du cycle, la phase asynchrone est modélisée par l'envoi d'un SoA puis la production ou la réception d'un ASnd. L'état AMN_TcycError est un état puit permettant de détecter une erreur de dépassement du temps de cycle dont la gestion n'est pas modélisée. Depuis les états AMN_*, en fonction des défaillances configurées, le nœud peut basculer à l'état RMN_Failing,
- au centre, ne participant pas aux échanges à l'état initial RMN_Monitoring_Activity et à l'état défaillant RMN_Failing. Ces états modélisent respectivement le nœud après

initialisation, observant les échanges pour décider de participer aux échanges en tant que AMN ou SMN, et l'état défaillant dans lequel le nœud ne réagit à aucun événement extérieur,

– participant aux échanges en tant que SMN dans les états SMN_*. La succession de ces états respecte le cycle EPL du point de vue SMN illustré dans la figure 2.2. Sur réception de l'information de début de cycle SoC, le nœud attend la réception d'un PReq pour produire son PRes. À la fin du cycle, sur réception du SoA, le nœud est susceptible de produire un ASnd s'il est désigné. Depuis les états SMN_*, en fonction des défaillances configurées, le nœud peut basculer à l'état RMN_Failing. Depuis les états SMN_Waiting_*, le nœud peut basculer à l'état AMN_Waiting_SoC en initiant un Active Managing Node Indication (AMNI) si aucune activité d'un nœud AMN n'a été observée sur le réseau depuis trop longtemps.

Description détaillée : Paramètres d'instanciation

Les paramètres de description des nœuds se résument à un recensement du nombre de nœuds (`NumberOfNodes`) et de leurs propriétés temporelles. Les propriétés suivantes constituent un sous-ensemble des paramètres de description d'équipement spécifiés par [76] et qui doivent accompagner tout équipement EPL.

`tSoC2PReq` donne le temps nécessaire à un nœud AMN pour envoyer une trame PReq après l'envoi du SoC,

`tPRes2PReq` donne le temps nécessaire à un nœud AMN pour envoyer un PReq après réception d'un PRes,

`tPRes2SoA` donne le temps nécessaire à un nœud AMN pour envoyer un SoA après réception d'un PRes ou envoi de son propre PRes,

`tSoA2ASnd` donne le temps nécessaire à un nœud AMN pour envoyer un ASnd après l'envoi du SoA, ou donne le temps nécessaire à un nœud SMN pour envoyer un ASnd après réception du SoA,

`tPReq2PRes` donne le temps nécessaire à un nœud SMN pour envoyer un PRes après réception du PReq.

Chacune de ces variables est un vecteur des valeurs pour l'ensemble des nœuds.

Les paramètres de configuration des nœuds du modèle constituent un sous-ensemble des paramètres de configuration EPL spécifiés par [76]. Ces paramètres sont destinés à donner une description commune de la configuration du cycle EPL.

`Tcyc` est le temps de cycle EPL (ou période entre deux SoC) fixé par l'utilisateur,

`PollProgSize` est le nombre de PRes planifiés dans la phase isochrone du cycle EPL. Pour l'exemple de la sous-section 3.3.1, le nombre de PRes sera compris entre 1 et 3 (le nombre de nœuds),

`PollProg` est le programme de consultation/production des PRes utilisé par l'AMN pour animer la phase isochrone. Quelle que soit sa position dans le programme, le PRes d'un nœud AMN sera toujours produit à la fin de la phase isochrone. Pour l'exemple de la sous-section 3.3.1, si tous les nœuds produisent un PRes (`PollProgSize=3`), la déclaration `PollProg=0,1,2` configure une phase isochrone EPL pour laquelle un PRes doit être produit par le nœud 0, puis le nœud 1, puis le nœud 2. Toutefois, si le nœud 0 est AMN, l'ordre sera alors 1→ 2→ 0,

`PResTimeout` est un vecteur regroupant le délai d'attente maximum de la réception du PRes pour chacun des nœuds. Il est utilisé par l'AMN pour ne pas attendre indéfiniment le PRes d'un nœud en bloquant le cycle EPL en cas de défaillance de ce dernier. Pour l'exemple de la sous-section 3.3.1, `PResTimeout` est un vecteur de dimension 3,

`ASndTimeout` est un vecteur regroupant le délai d'attente maximum de la réception d'un ASnd pour chacun des nœuds. Il est utilisé par l'AMN pour ne pas attendre indéfiniment un ASnd en bloquant le cycle EPL en cas de défaillance du nœud invité à produire l'ASnd. Pour l'exemple de la sous-section 3.3.1, `ASndTimeout` est un vecteur de dimension 3.

En plus de ces paramètres, chaque modèle de nœud est instancié en donnant une valeur aux variables suivantes :

`idNode` un identifiant unique, compris entre zéro et le nombre total de nœuds moins un,

`TSwitchOver` donne la durée maximale au cours de laquelle aucune activité d'un AMN n'a été détectée et après laquelle le nœud considéré bascule de l'état SMN à l'état AMN. Ce délai est unique pour chaque nœud afin d'éviter que plusieurs basculent vers le comportement AMN en même temps,

`TAfterSwitchOver` donne un délai d'attente après basculement. Après détection de la disparition du nœud AMN et basculement à l'état AMN, il est possible de configurer le délai d'attente du nœud entre l'envoi de la trame AMNI et l'envoi de son premier SoC. Ce délai peut permettre d'imposer un temps minimal entre les deux trames, mais aussi de conserver la régularité du SoC du point de vue de la cellule.

Les paramètres de défaillance doivent être choisis par l'utilisateur pour définir les conditions pour lesquelles les propriétés attendues seront vérifiées par le modèle d'architecture. Plus les possibilités de défaillance sont élargies, plus la vérification du modèle aura de chances d'atteindre les limites de mémoire au niveau du calculateur ou dépasser les limites de temps supportées par l'utilisateur.

`NodesFailuresMax` configure le nombre autorisé de nœuds simultanément défaillants,

`AMNFailuresMax` configure le nombre autorisé de nœuds simultanément défaillants depuis le comportement AMN,

`SMNFailuresMax` n'est pas configuré par l'utilisateur mais calcule le nombre autorisé de nœuds simultanément défaillants depuis le comportement SMN à partir de NodesFailuresMax et AMNFailuresMax,

`NodeFixingAllowed` autorise ou non la modélisation de la réparation des nœuds, c'est-à-dire leur retour à la participation aux échanges, après un passage à l'état défaillant. En raison de l'indéterminisme introduit au niveau de la transition modélisant la réparation dans l'automate Uppaal du nœud, la vérification d'une configuration pour laquelle `NodeFixingAllowed=1` inclut la vérification d'une configuration similaire, mais pour laquelle `NodeFixingAllowed=0`,

`AllStatesAMNFailuresEnabled` autorise ou non la défaillance du modèle à états du nœud AMN depuis tous ses états. Ce paramètre permet d'inhiber certains franchissements des transition pour limiter les séquences possibles,

`AllStatesSMNFailuresEnabled` autorise ou non la défaillance du modèle à états du nœud SMN depuis tous ses états. Ce paramètre permet d'inhiber certains franchissements des transition pour limiter les séquences possibles,

`NumberOfAsndEnabled`, `NumberOfAsndEnabled` et `ASndSenderIDLimiter` permettent de réduire l'indéterminisme du modèle en limitant le nombre de producteurs possible de la trame ASnd à la liste identifiée par le vecteur `ASndSenderIDLimiter`.

Description détaillée : Automate Uppaal

La figure 3.21 donne la représentation de l'automate UPPAAL associé à la classe nœud. Une vue répartie sur deux pages est donnée en annexe 4.4.

Animation des échanges en tant que AMN :

Depuis l'état initial RMN_Monitoring_Activity, l'invariant associé à l'état initial de l'automate entraîne le franchissement de la transition vers le comportement AMN à l'échéance du délai de basculement `tSwitchOver`.

L'animation des échanges va assurer l'enchaînement correct des différentes phases du cycle EPL (Départ cycle, isochrone, asynchrone, inactive, cf. 2.1.2).

L'initiation régulière d'une synchronisation datagramme de type SoC est assurée depuis l'état AMN_Waiting_SoC par son invariant et la franchissabilité simultanée de la transition vers l'état AMN_Delaying_PReq.

L'activation des états AMN_Delaying_PReq, AMN_Waiting_PRes et AMN_Delaying_PRes correspond à la phase isochrone du cycle EPL. Depuis l'état AMN_Delaying_PReq, l'AMN va consulter un à un les nœuds SMN dans l'ordre planifié par le vecteur PollProg (déclaré en tant que variable globale, donc commun à tous les nœuds).

Le modèle du nœud envoie un datagramme de type PReq avant de passer à l'état AMN_Waiting_PRes. L'évolution depuis l'état AMN_Waiting_PRes est provoquée par la réception d'un datagramme de type PRes provenant du nœud consulté (ExpectedPRes()) ou par l'échéance du délai `PResTimeout`. Le passage à l'état AMN_Delaying_PReq, AMN_Delaying_PRes ou AMN_Delaying_SoA dépend du rang atteint dans le vecteur PollProg. À l'issue du programme, la méthode `nextIsAMNPRes()` indique si le nœud jouant le rôle d'AMN est planifié dans le programme. Si la condition est vérifiée, un datagramme PRes est généré après attente dans l'état AMN_Delaying_PRes. Si le nœud jouant le rôle d'AMN n'est pas planifié dans le programme, la transition vers l'état AMN_Delaying_SoA est directement franchie.

La modélisation de la phase isochrone du cycle EPL se termine par l'attente dans l'état AMN_Delaying_SoA. Ce délai modélise les caractéristiques de réactivité du nœud.

L'activation des états AMN_Waiting_Or_Delaying_ASnd et AMN_Waiting_SoC correspond à la phase asynchrone du cycle EPL. Depuis l'état AMN_Delaying_SoA, le nœud envoit un datagramme SoA et passe à l'état AMN_Waiting_Or_Delaying_ASnd. Au cours du franchissement de la transition, le producteur de la prochaine trame ASnd est choisi aléatoirement par l'intermédiaire de la sélection `randID:idNode` et de l'action `ASndID:=randID`. Si le nœud choisi est l'AMN, l'envoi d'un datagramme ASnd est retardé depuis l'état courant avant passage à l'état AMN_Waiting_SoC. Sinon, la transition vers l'état AMN_Waiting_SoC sans envoi de datagramme est franchie à la réception d'une trame ASnd ou à l'échéance du délai `ASndTimeout`. Si depuis l'état AMN_Waiting_Or_Delaying_ASnd la durée du cycle EPL est atteinte, aucune des deux dernières transitions ne peut être franchie et l'automate bascule dans l'état puit AMN_TcycError qui permettra de signaler une incompatibilité entre paramètres temporels saisis et réactivité désirée.

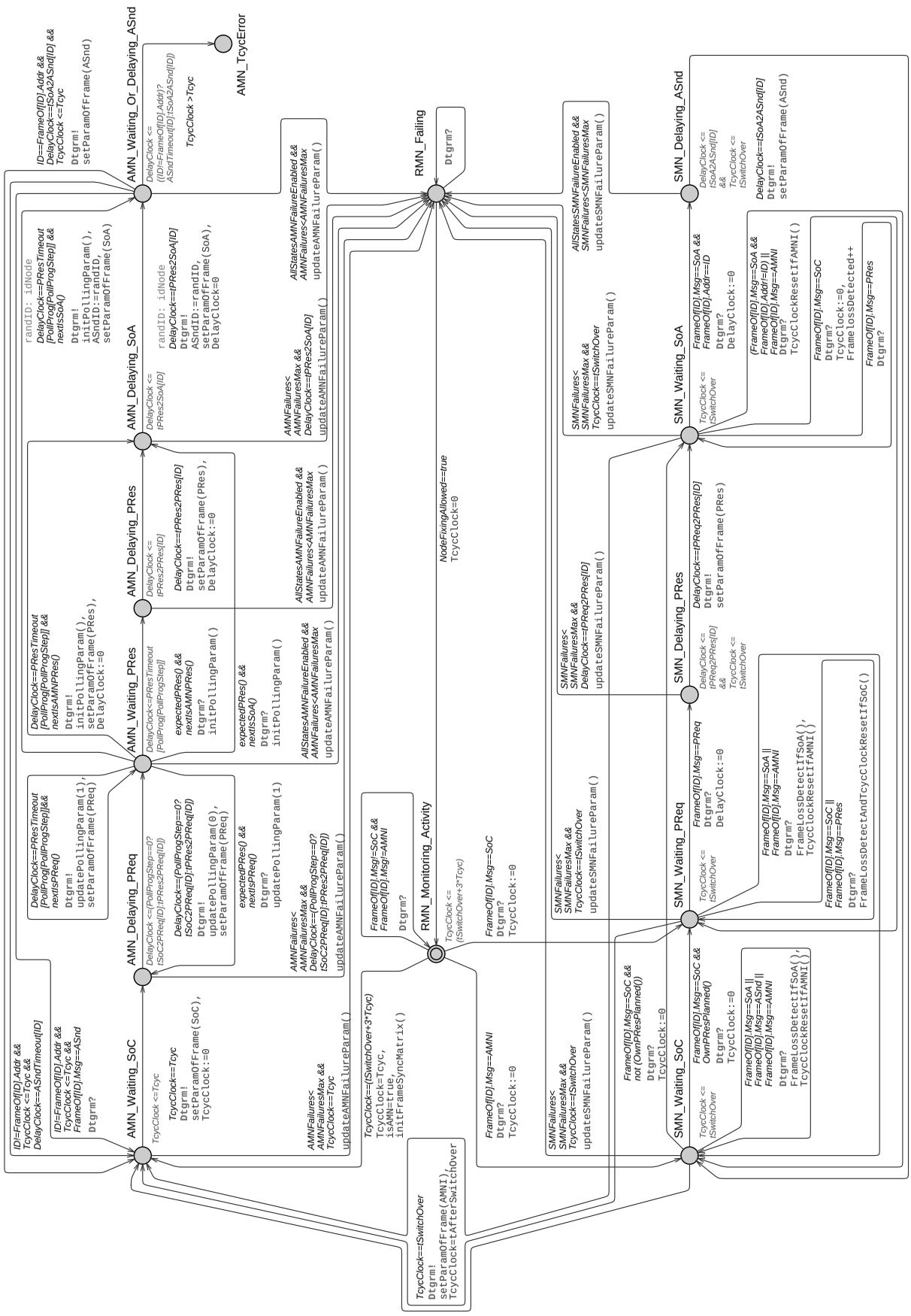


FIGURE 3.21 – Modèle à états UPPAAL d'un nœud.

Participation aux échanges en tant que SMN :

Depuis l'état initial RMN_Monitoring_Activity, la transition vers le comportement SMN est franchie sur réception d'un datagramme envoyé par un AMN (SoC ou AMNI).

Comme pour la branche AMN, les différents états SMN_* peuvent être associés aux trois phases du cycle EPL.

La réception d'un SoC représente la phase de départ cycle et le début de la phase isochrone correspondant à l'activation des états SMN_Waiting_PReq, SMN_Delaying_PRes et SMN_Waiting_SoA. Depuis l'état SMN_Waiting_PReq, à la réception d'un PReq qui lui est destiné (*FrameOf[ID].Msg==PReq*), l'automate bascule à l'état SMN_Delaying_PRes afin de retarder l'envoi du datagramme de réponse PRes. Une fois ce délai introduit, la transition vers l'état SMN_Waiting_SoA d'attente du signal de début de la phase asynchrone. L'activation des états SMN_Delaying_ASnd et SMN_Waiting_SoC correspond à la phase asynchrone du cycle EPL. Depuis l'état SMN_Waiting_SoA, à la réception d'un datagramme SoA, la transition vers l'état SMN_Delaying_ASnd est franchie si le nœud est désigné comme le producteur du datagramme (*FrameOf[ID].Addr==ID*). L'envoi du datagramme ASnd est alors retardé durant l'activation de l'état SMN_Delaying_ASnd. Si le nœud n'est pas désigné comme le producteur du datagramme (*FrameOf[ID].Addr!=ID*), l'automate passe directement à l'état SMN_Waiting_SoC.

La succession des états décrits ci-dessus correspond à un enchaînement normal des échanges. Si depuis un des états SMN_*, la trame reçue ne correspond pas à la vision de l'automate de l'état d'avancement du cycle EPL, un compteur de perte de trame (`FramelossDetected`) est incrémenté, et l'automate modifie l'état actif.

La modélisation de la redondance logicielle décrite dans la section 2.2 intervient à partir des états d'activation SMN_Waiting_SoC,SMN_Waiting_PReq et SMN_Waiting_SoA. Depuis ces états, si l'automate n'a reçu aucune trame initiée par un AMN pendant la durée de basculement configurée, la condition de transition *TcycClock==tSwitchOver* est vérifiée. La transition vers l'état AMN_Waiting_SoC est alors franchie le datagramme AMNI est émis.

Modélisation de la défaillance du nœud :

Depuis les états AMN_* et SMN_*, un nœud peut basculer de façon indéterminée vers l'état défaillant RMN_Failling tant que le nombre maximal de défaillances de nœuds respectivement AMN ou SMN n'a pas été atteint. Depuis cet état le nœud continue de recevoir les datagrammes envoyés par le Link Selector, mais il ne participe pas aux échanges.

Depuis l'état RMN_Failing, la réparation du nœud correspond à la transition vers l'état RMN_Monitoring_Activity. Le franchissement de cette transition est conditionné par la valeur du paramètre `NodeFixingAllowed` du modèle. L'indéterminisme de franchissement d'une transition validée (sous section 3.2.1) est utilisé pour modéliser la réparation du nœud à n'importe quel moment après basculement à l'état défaillant RMN_Failing. Après retour à l'état RMN_Monitoring_Activity, l'automate observe les trames reçues suffisamment longtemps (3 cycles EPL plus le délai configuré de basculement) pour décider de franchir une transition vers un comportement AMN ou SMN.

3.4.2 Modèle d'un Link Selector

Description informelle

Dans la sous-section 3.3.3, nous avons rappelé les tâches incombant à la classe LS, la duplication de trame sortante et sélection de trames entrantes au niveau du nœud. L'automate UPPAAL représenté dans la figure 3.22 modélise le comportement du LS. La boucle Idle↷Waiting_Redundant_Frame↷Sending_Datagram↷Idle modélise la fonction de sélection de

trames entrantes à transmettre aux couches supérieures. S'il n'y a pas correspondance entre deux trames censées être redondantes, l'automate bascule à l'état puit RedundantFrameMismatchIssue pour signalisation. La gestion de ce type d'erreur n'est pas prise en compte par le modèle. La boucle Idle↷Sending_Frame↷Idle modélise la fonction de duplication de trame sortante sur réception d'un datagramme depuis les couches supérieures. L'émission est simultanée sur les deux ports, mais un délai d'envoi variable `NodeJitter[NodeID]` est introduit.

Description détaillée : paramètres d'instanciation

Seules les définitions de paramètres de description suffisent à définir un LS. Le modèle de LS utilise également un paramètre décrivant le couple (nœud ; LS) :

`NodeID` est la variable d'instanciation rappelant l'identifiant du nœud desservi,

`LSPortsAssignement` est la matrice correspondant au tableau 3.1. Elle modélise les connexions physiques entre LS et IDS. Elle est utilisée par des gardes de transition. Le franchissement de transition sur réception d'un événement trame est conditionné par la valeur de ses champs,

`NodeJitter` est un vecteur regroupant la variation de délai d'émission de trame par chacun des couples (nœud ; LS). Cette variation regroupe les variations internes possibles dues aux cycles internes (traitements ou communications sur le fond de panier).

Le modèle d'un LS n'utilise ni paramètres correspondant à une configuration des fonctions remplies ni paramètres servant à modéliser une défaillance.

Description détaillée : automate Uppaal

L'automate Uppaal modélisant le comportement du LS est illustré dans la figure 3.22.

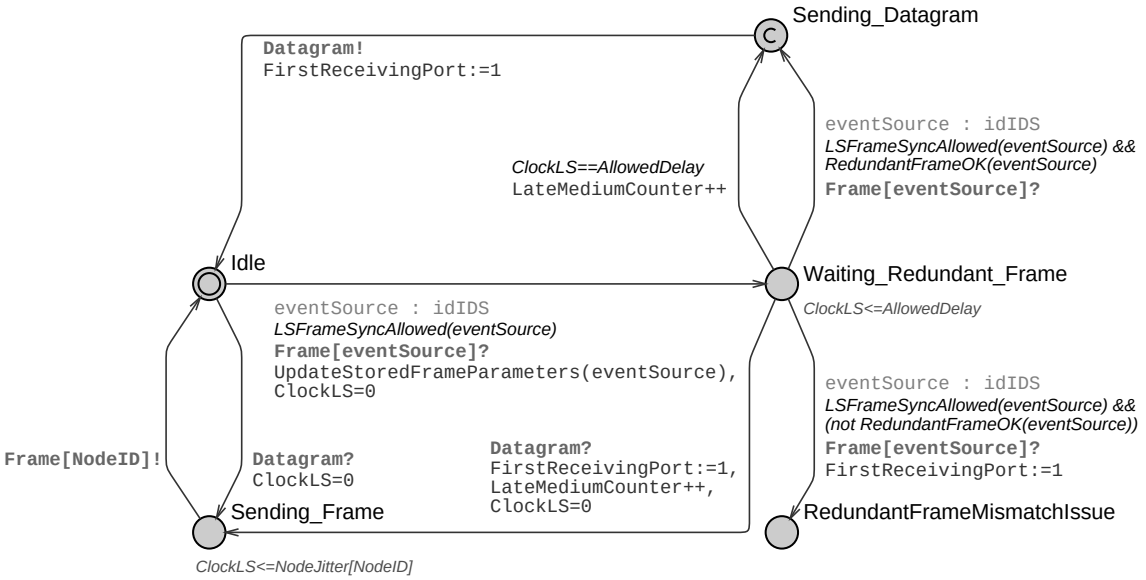


FIGURE 3.22 – Modèle à états Uppaal d'un Link Selector.

Depuis l'état Idle, à la réception d'un événement `Datagram` émis par le modèle de nœud desservi, la transition vers l'état Sending_Frame est franchie. La transition de sortie

est franchie entre un temps nul et un délai fixé par le paramètre `NodeJitter` du nœud desservi. La duplication de trame est modélisée par la synchronisation `Frame[NodeID]!` qui doit être reçue simultanément sur chaque IDS connectée.

Depuis l'état Idle, le champ de sélection `eventSource :idIDS` rend le modèle potentiellement capable de synchroniser le franchissement de la transition vers l'état Waiting_Redundant_Frame avec n'importe quel événement trame (`Frame[eventsource]`) émis par un automate UPPAAL du modèle d'architecture. La garde *LSFrameSyncAllowed(eventsource)* permet de limiter le franchissement de la transition aux événements pertinents. La fonction `LSFrameSyncAllowed()` renvoie un résultat binaire à partir des informations liées à la trame et de la matrice `LSPortsAssignement`.

Depuis l'état Waiting_Redundant_Frame, nous avons modélisé le mode de sélection de trame le plus simple. Un datagramme est envoyé vers l'automate UPPAAL du nœud après réception de la trame redondante ou après un délai d'attente.

- Si aucun événement n'est reçu à l'issue de la valeur de temporisation `AllowedDelay` (*ClockLS==AllowedDelay*), la transition vers l'état Sending_Datagram est franchie et le compteur `LateMediumCounter` est incrémenté. L'incrémentation du compteur est considérée comme la détection d'une défaillance sur un médium par le LS.
- Si la prochaine trame reçue est la trame redondante, *RedundantFrameOK(eventSource)* est vrai, la transition vers l'état Sending_Datagram est franchie.
- Si la prochaine trame reçue n'est pas la trame redondante, on vérifie la condition sur la garde*not RedundantFrameOK(eventSource)*, la transition vers l'état puit RedundantFrameMismatchIssue est franchie afin de bloquer l'évolution de l'automate. Cet état puit permet de détecter le cas de trafics différents entre les deux média dont les conséquences ne sont pas gérées par la modélisation.
- Si un datagramme est reçu depuis l'automate UPPAAL du nœud, il est symptomatique de deux situations.
  Soit une défaillance de médium entraîne un ralentissement trop important des communications. Le cycle EPL ne peut se compléter dans le temps `Tcyc` imparti. Soit le nœud possède des paramètres ou une configuration temporelle qui le rendent trop réactif par rapport à l'ensemble du réseau. Du point de vue LS, le franchissement de cette transition entraîne la détection d'une défaillance sur un médium et annule le traitement de la trame reçue sans interruption de l'évolution du modèle. Si la première situation est à l'origine de la synchronisation, elle sera détectée au niveau du nœud. Si la seconde situation est à l'origine de la synchronisation, elle entraînera un problème de collision détecté au niveau du médium.

Depuis l'état (commited) Sending_Datagram, la transition vers l'état Idle est franchie à temps nul en envoyant un datagramme au nœud desservi.

3.4.3 Modèle d'une partie d'un médium, l'IDS

Description informelle

Dans la sous-section 3.3.2, nous avons rappelé les tâches incombant à la classe IDS, la transmission d'une trame reçue avec l'introduction d'un délai ainsi que la modélisation d'une défaillance temporaire ou définitive empêchant cette transmission. L'automate UPPAAL représenté dans la figure 3.23 modélise le comportement d'une IDS. La boucle Idle_Or_Failling↷Delaying_Frame↷Idle_Or_Failling modélise la réception d'une trame et l'attente avant sa transmission. Le basculement à l'état puit FrameCollisionOrConfigurationIssue permet de détecter des erreurs de collision au niveau de l'IDS (l'IDS reçoit une nouvelle trame

alors qu'il n'a pas encore transmis la précédente trame reçue) ou de découpage du médium en IDS suffisamment fin. La gestion de la première erreur n'est pas prise en compte par le modèle. Le modèle gère sa possible défaillance et réparation par l'intermédiaire des transitions bouclant sur l'état Idle_Or_Failling.

Description détaillée : paramètres d'instanciation

Les paramètres de description des IDS se résument à un recensement du nombre d'IDS (`NumberOfIDS`) et de leurs propriétés temporelles.

`idIDS` est un identifiant unique différent des identifiants de nœud. Pour le découpage choisi par la figure 3.13, les identifiants suivent ceux des nœuds. Ils sont compris entre 3 et 8, et sont répartis par l'utilisateur.

`IDSLatency` donne la latence moyenne introduite par chacune des IDS. Les valeurs sont déduites des caractéristiques des composants d'interconnexion constituant les IDS. Pour le découpage choisi par la figure 3.13, `IDSLatency` est un vecteur de dimension le nombre d'IDS pour lequel toutes les valeurs sont égales à la latence moyenne introduite par le concentrateur.

`IDSJitter` donne la gigue introduite par chacune des IDS. Les valeurs sont également déduites des caractéristiques des composants d'interconnexion constituant les IDS. Pour le découpage choisi par la figure 3.13, `IDSJitter` est un vecteur de dimension le nombre d'IDS pour lequel toutes les valeurs sont égales à la gigue introduite par le concentrateur.

`LinksBetweenIDS` la matrice correspondant au tableau 3.2. Elle modélise les connexions physiques entre IDS. Elle est utilisée par des gardes de transition. Le franchissement de transition sur réception d'un événement trame est conditionné par la valeur de ses champs.

Les paramètres de défaillance traduisent les deux questions que nous nous sommes posées sur les conséquences d'une défaillance de médium.

`IDSFailure` autorise l'occurrence d'une et une seule défaillance d'IDS. En effet, nous ne voulons vérifier la tolérance qu'à un défaut médium,

`IDSFixingAllowed` autorise la réparation de l'IDS défaillant pour assurer de nouveau le transfert d'événement. Nous voulons vérifier de cette manière que, par exemple, le remplacement d'un câble sectionné, ne peut perturber le cycle des échanges EPL ou la redondance logicielle. De par l'indéterminisme introduit au niveau de la transition modélisant la réparation dans l'automate Uppaal de l'IDS, la vérification d'une configuration pour laquelle `IDSFixingAllowed=1` inclut la vérification de la même configuration pour laquelle `IDSFixingAllowed=0`.

Le modèle d'une IDS n'utilise pas de paramètres correspondant à une configuration des équipements d'interconnexion modélisés.

Description détaillée : automate Uppaal

Depuis l'état inactif non défaillant Idle_Or_Failling avec `Failling=False`, le champ de sélection `eventSource:idNode8IDS` rend le modèle potentiellement capable de synchroniser le franchissement de la transition vers l'état Delaying_Frame avec n'importe quelle

réception de trame (événement `Frame[eventsource]`). Le franchissement de la transition est limité aux trames pertinentes grâce à la condition *IDSFrameSyncAllowed(eventSource)* sur la garde. Comme pour le LS, ce choix de modélisation nous permet de décrire un filtre des trames dites pertinentes par les variables globales appelées `LinksBetweenIDS` et `NodePortsAssignement`. Celles-ci sont utilisées par la méthode `IDSFrameSyncAllowed()`. L'objectif est également de garder un modèle d'IDS commun à n'importe quelle description d'architecture.

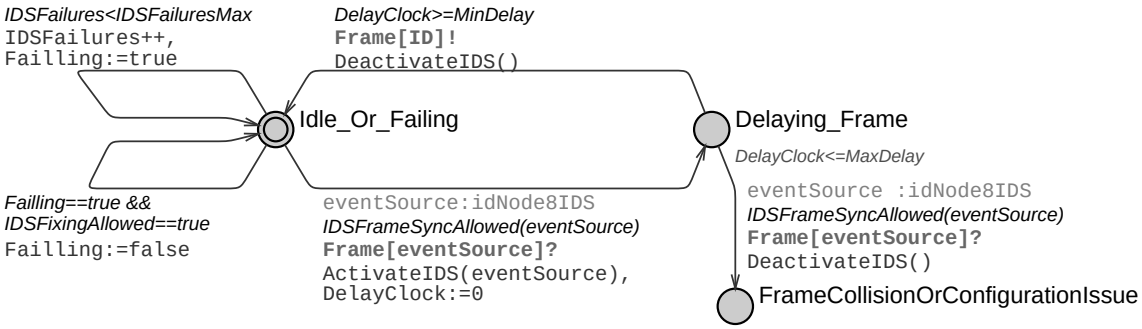


FIGURE 3.23 – Modèle à états UPPAAL d'une Sphère Isochrone de Diffusion.

Le franchissement de la transition vers l'état Delaying_Frame s'accompagne de l'activation de l'IDS avec l'action `ActivateIDS(eventSource)`. La méthode `ActivateIDS()` modifie la variable de filtre `ListensToIDS` pour empêcher la repropagation en amont d'une trame.

Depuis l'état Delaying_Frame, la transmission d'une trame reçue est retardée d'une période comprise entre `MinDelay` et `MaxDelay`, correspondant respectivement aux valeurs de $Delai + \frac{gigue}{2}$ et $Delai - \frac{gigue}{2}$. Par conséquent, le délai modélisé a une distribution constante. Dans la réalité le délai de transmission aura une distribution non uniforme, comme nous pouvons le voir dans [19]. Toutefois, la distribution constante convient bien à la vérification de propriété jusqu'aux conditions limites.

Le franchissement de la transition vers l'état Idle_Or_Failling s'accompagne de la transition de l'événement par l'émission d'une trame (`Frame[ID]!`) et la désactivation de l'IDS par l'action `DeactivateIDS()`. La méthode `DeactivateIDS()` rétablit les valeurs modifiées lors de l'activation.

Lorsque l'automate est à l'état Delaying_Frame de temporisation de la transmission d'événement, toute trame reçue le fera basculer dans l'état puit FrameCollisionOrConfigurationIssue. En effet, une telle situation correspond soit à la circulation simultanée de deux trames dans le sous-domaine de diffusion, soit à l'occurrence d'une collision ou de deux trames qui se suivent de trop près pour la paramétrisation de l'IDS. Les deux situations ne sont pas gérées par la modélisation. La première doit être détectée pour vérifier des propriétés de l'architecture. La seconde signale une erreur de découpage du médium lors de la modélisation de l'architecture. Un redécoupage de l'IDS incriminé suffit à obtenir une description qui ne génèrera plus l'erreur.

Depuis l'état Idle_Or_Failling avec`Failling=false`, si la défaillance d'un IDS est autorisée par la description de l'architecture (*IDSFailuresEnabled*), la transition mettant à jour la variable `Failling` avec l'action `Failling:=true` peut être franchie de façon indéterminée. L'IDS ne participera plus alors à la transmission de trame et restera à l'état Idle_Or_Failling.

Depuis l'état Idle_Or_Failling avec`Failling=true`, si la réparation de l'IDS défaillant est autorisée par le paramètre (*IDSFixingAllowed==true*), la transition mettant à jour la variable

`Failling` avec l'action `Failling:=false` peut être franchie de façon indéterminée.

La modélisation autorise au plus une défaillance d'une IDS et son éventuelle réparation.

3.5 Modélisation des propriétés attendues

La modélisation d'architecture proposée dans la section précédente fournit les variables nécessaires à l'identification des états considérés comme indésirables du système décrit. Ces variables nous servent à formuler les propriétés énoncées dans la section 2.4 en logique temporelle TCTL [65] sous la forme indiquée dans la sous-section 3.2.4. Les formules suivantes décrivant les propriétés ne dépendent pas du paramétrage du modèle.

P1 : Un seul nœud maître des échanges à la fois Pour un état de contrôle, l'expression `exists (i:idNode) (exists (j:idNode)(i!=j & & RMN(i).isAMN==true & & RMN(j).isAMN==true)` sera vraie s'il existe deux identifiants de nœuds différents pour lesquels les automates UPPAAL associés peuvent avoir leur variable privée `isAMN` vraie en même temps.

P2 : Il n'y a pas de collision provoquée sur le réseau Pour un état de contrôle, l'expression `exists (i:idIDS) (IDS(i).FrameCollisionOrConfigurationIssue==1)` sera vraie s'il existe un identifiant d'IDS pour lequel l'automate UPPAAL associé a son état FrameCollisionOrConfigurationIssue actif.

P3 : Il n'y a pas de dépassement possible du temps de cycle configuré Pour un état de contrôle, l'expression `(exists (i:idNode) (RMN(i).AMN_ TcycError==1))` sera vraie s'il existe un identifiant de nœud pour lequel l'automate UPPAAL a son état AMN_ TcycError actif.

P4 : Il n'y a pas de décalage de trafic entre chaque médium Pour un état de contrôle, l'expression `exists (i:idNode) (LS(i).RedundantFrameMismatchIssue==1)` sera vraie s'il existe un identifiant de nœud pour lequel l'automate UPPAAL associé a son état RedundantFrameMismatchIssue actif.

P5 : Il n'y a pas de perte de trame La perte de trame surveillée est la perte de trame du point de vue de la logique du protocole, c'est-à-dire si la succession des trames reçues ne respecte pas la succession spécifiée. La perte d'une trame dûe à une défaillance de médium ne sera pas détectée si la succession spécifiée est respectée. Pour un état de contrôle, l'expression `exists (i:idNode) (RMN(i).FramelossDetected!=0)` sera vraie s'il existe un identifiant de nœud pour lequel l'automate UPPAAL associé a sa variable privée `FramelossDetected` différente de zéro.

P6 : Il n'y a pas de bloquage du modèle Cette expression sera vraie si un des automates passe dans un de ses états puits de détection d'erreur ; elle comprend certaines des propriétés précédentes. Par contre, si cette expression est vraie sans qu'un des états puits se soit actif, cela signifie que la modélisation n'a pas pris en compte la situation particulière liée à cet état de contrôle ou que le paramétrage n'est pas pertinent. Par conséquent, cette propriété n'apporte pas de vérification sur la solution de redondance, mais elle permet de contrôler la pertinence du modèle pour une configuration choisie.

Nous cherchons à vérifier qu'aucune évolution ne conduit à valider une des formules. C'est pourquoi l'expression réellement vérifiée définit la non-occurrence de l'union de ces états de contrôle indésirables. Les propriétés attendues pour une architecture modélisée seront vérifiées si l'expression $A[]\ not\ (P1\ \&\&\ P2\ \&\&\ P3\ \&\&\ P4\ \&\&\ P5)$ est vérifiée.

3.6 Bilan sur le modèle

3.6.1 Méthode d'instanciation

L'instanciation d'une architecture EPL-HA à l'aide du modèle générique UPPAAL proposé consiste à renseigner les valeurs des paramètres de réglage représentés dans la figure 3.24. Vingt-quatre paramètres ou groupes de paramètres de réglage sont nécessaires à la description d'un modèle particulier.

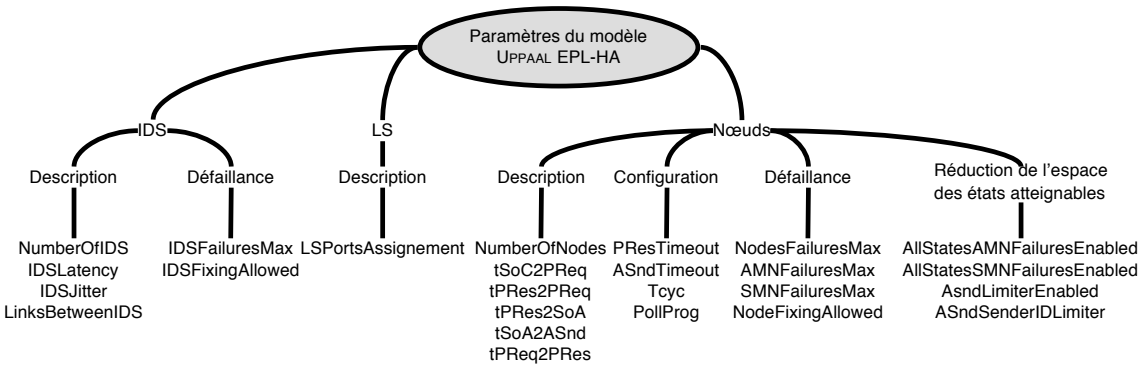


FIGURE 3.24 – Paramètres de réglage du modèle d'architecture EPL-HA.

Nous appliquons la méthode à l'architecture représentée sur la figure 3.25. Ce type d'architecture peut être envisagé dans le cas de cellules étendues sur de longues distances. Cet étirement de la cellule permet de mettre en œuvre des traitements locaux dans les contrôleurs de terrain (au plus près du processus) ainsi que des traitements centralisés en tête de cellule (et/ou des remontées d'informations vers la supervision).

Cette architecture utilise une topologie en étoile qui connecte 4 nœuds : un contrôleur de cellule et 3 nœuds distants. La particularité de la topologie considérée est qu'une des branches (entre le contrôleur de cellule et les trois nœuds) est relativement longue. C'est pourquoi des convertisseurs Cuivre/Fibre sont utilisés entre les deux îlots.

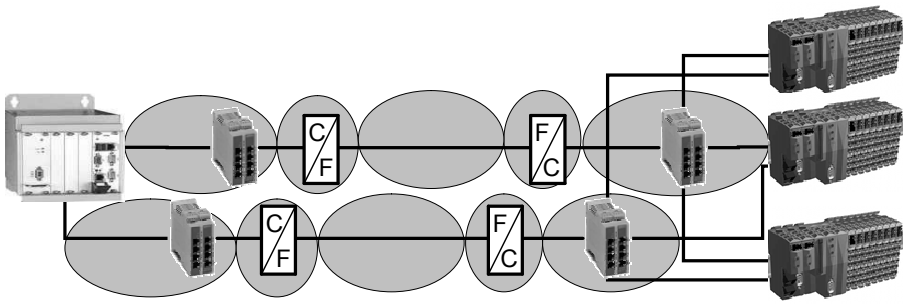


FIGURE 3.25 – Architecture considérée.

Les premiers paramètres qui peuvent être définis concernent les nœuds. Les branches paramètres de description des nœuds et paramètres de configuration des nœuds sont renseignées par les origines suivantes :

a. les valeurs fournies par les constructeurs des nœuds modélisés,
b. le calcul manuel des délais de propagation. Il s'agit de déterminer le délai maximum entre deux nœuds en additionnant les délais introduits par les composants traversés,
c. l'utilisation des résultats de calculs effectués par un logiciel dédié à la conception d'architectures EPL. Par exemple, l'application EPLDesigner (Alstom Power), dont une copie d'écran est donnée en annexe 4.4, permet de remplacer le calcul manuel pour définir une partie des paramètres de réglage des nœuds,
d. la spécification de contraintes par l'utilisateur. Par exemple, ce dernier peut vouloir imposer un temps de cycle pour répondre aux besoins du processus piloté. La vérification permettra alors de valider la compatibilité de l'architecture imaginée avec le temps de cycle désiré.

Le tableau 3.3 donne, pour chaque paramètre de description et de configuration, quelle(s) origine(s) permet(tent) de définir sa valeur ainsi que la valeur retenue pour notre exemple.

Description des nœuds	Source	Exemple de la figure 3.25
`NumberOfNodes`		4
`tSoC2PReq` x100ns	a	{300,300,300,300}
`tPRes2PReq` x100ns	a	{300,300,300,300}
`tPRes2SoA` x100ns	a	{300,300,300,300}
`tSoA2ASnd` x100ns	a	{300,300,300,300}
`tPReq2PRes` x100ns	a	{60,60,60,60}
Configuration des nœuds		
`PResTimeout` x100ns	b, c	{550,550,550,550}
`ASndTimeout` x100ns	b, c	{800,800,800,800}
`Tcyc` x100ns	c, d	10000
`PollProg`	c, d	{0,1,2,3}

TABLE 3.3 – Paramètres de description et de configuration des nœuds

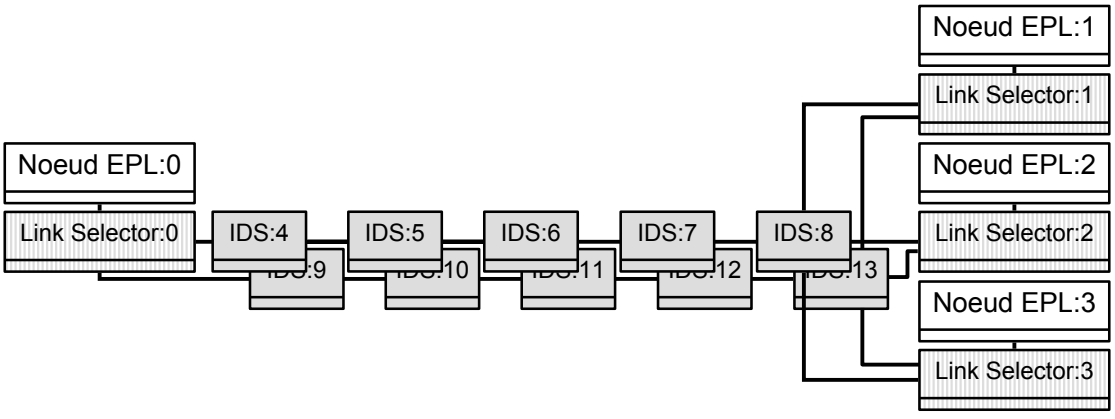


FIGURE 3.26 – Diagramme d'objet modélisant l'architecture considérée.

La seconde étape est de définir un découpage des domaines de diffusion en IDS. Il faut choisir un rayon maximum, c'est-à-dire un délai maximum introduit par les composants regroupés dans une IDS. Celui-ci doit être inférieur à la valeur minimale renseignée dans les vecteurs `tSoC2PReq`, `tPRes2PReq`, `tPRes2SoA`, `tSoA2ASnd` et `tPReq2PRes` pour que le modèle soit vérifiable.

Dans la figure 3.25, le découpage en IDS résultant est déjà représenté. Il conduit au diagramme d'objet de la figure 3.26 pour la modélisation (les échanges ne sont pas représentés). Le modèle retenu est constitué de 4 instances des classes nœuds et LS. Chaque médium est découpé en quatre instances d'IDS.

À partir des données sur les composants d'interconnexion et des choix de topologie de l'utilisateur, il est possible de renseigner les paramètres de description des IDS et des LS. Le tableau 3.4 donne les valeurs retenues pour notre exemple.

Description des IDS	Exemple de la figure 3.25
`NumberOfIDS`	10
`IDSLatency` x100ns	{10,50,45,50,10,10,50,20,50,10}
`IDSJitter` x100ns	{2,2,0,2,2,2,2,0,2,2}
`LinksBetweenIDS`	{{0,1,0,0,0,0,0,0,0,0}, {1,0,1,0,0,0,0,0,0,0}, {0,1,0,1,0,0,0,0,0,0}, {0,0,1,0,1,0,0,0,0,0}, {0,0,0,1,0,0,0,0,0,0}, {0,0,0,0,0,0,2,0,0,0}, {0,0,0,0,0,2,0,2,0,0}, {0,0,0,0,0,0,2,0,2,0}, {0,0,0,0,0,0,0,2,0,2}, {0,0,0,0,0,0,0,0,2,0}}
Description des LS	
`LSPortsAssignment`	{{1,0,0,0,0,2,0,0,0,0}, {0,0,0,0,1,0,0,0,0,2}, {0,0,0,0,1,0,0,0,0,2}, {0,0,0,0,1,0,0,0,0,2}} ;

TABLE 3.4 – Paramètres de description des IDS et des LS

Finalement, l'utilisateur doit préciser les défaillances qu'il souhaite étudier ainsi que les valeurs des paramètres permettant de réduire l'espace des états atteignables :

- la défaillance d'une IDS ainsi que la réparation de celle-ci,
- le nombre de nœuds simultanément à l'état défaillant. Il faut également préciser le nombre de nœuds passant à l'état défaillant depuis un des états Active Managing Node. Le nombre de nœuds passant à l'état défaillant depuis un des états Stand-by Managing Node sera déduit de ces deux dernières valeurs. Enfin, il faut indiquer si la réparation des nœuds défaillants est possible,
- la limitation des nœuds susceptibles de produire une trame asynchrone ASnd. Il faut alors donner la liste des nœuds,
- la limitation des états à partir desquels un nœud peut passer à l'état défaillant.

Le tableau 3.5 donne les valeurs de ces derniers paramètres de réglage pour des évolutions sans défaillance d'IDS, avec une unique défaillance de nœuds AMN possible et réparable. Les nœuds 0 et 1 sont les seuls à produire une trame ASnd et les états

d'origine de la défaillance des nœuds sont limités.

Défaillance et abstraction	Exemple pour la figure 3.25
`IDSFailureEnabled`	false
`IDSFixingAllowed`	false
`NodesFailuresMax`	1
`AMNFailuresMax`	1
`NodeFixingAllowed`	true
`AllStatesAMNFailuresEnabled`	false
`AllStatesSMNFailuresEnabled`	false
`AsndLimiterEnabled`	true
`AsndSenderIDLimiter`	{0,1}

TABLE 3.5 – Paramètres de défaillance et de réduction de l'espace des états atteignables

Les propriétés à vérifier sont formulées pour s'adapter aux paramètres de réglage. L'utilisateur n'a donc pas besoin de modifier les formules logiques énoncées dans la section 3.5.

3.6.2 Choix d'abstraction

Pour parvenir au modèle générique d'architecture présenté dans ce chapitre, différents choix d'abstraction ont été faits. Le but est de limiter l'espace des états atteignables par le modèle pour qu'il soit vérifiable, c'est-à-dire lutter contre l'explosion combinatoire, sans filtrer des états intéressants en regard des propriétés à vérifier. La description du modèle générique nous a permis de mettre en évidence nos différents choix d'abstraction.

Plutôt que définir un modèle associant un sous-système à chaque composant du réseau, nous avons défini la classe de sous-système IDS qui permet de modéliser un groupe de composants d'interconnexion d'un domaine de diffusion. Cette abstraction participe non-seulement à la limitation des comportements indépendants dans un modèle d'architecture mais aussi à la généricité du modèle qui ne dépend pas du type de composants d'interconnexion utilisés. La généricité profite également de l'utilisation des paramètres de description d'un nœud spécifiés par [76]. La classe de sous-système nœud permet ainsi de modéliser toute sorte d'équipement EPL existant ou en développement.

La compression du temps introduit par les trames nous permet de réduire la combinatoire qu'occasionnerait la gestion de dates de début et de fin de trame. Ceci est possible car les délais introduits par les trames, les composants regroupés en IDS ou les nœuds sont indépendants. Les premiers dépendent des quantités de données encapsulées et du débit, les autres sont des propriétés intrinsèques des équipements. Cette indépendance ne serait plus vraie dès lors que l'on utiliserait des réseaux commutés.

Une conséquence de cette abstraction est que l'on doit tenir compte de la compression du temps pour paramétrer le temps de cycle EPL. Il faut réduire le cycle désiré d'autant que les délais pris par les trames. Cette réduction implique une diminution des intervalles de basculement pour la redondance logicielle. Toutefois, nous n'avons pas constaté que cette conséquence conduisait à rendre une architecture non valide sur les cas d'architecture vérifiés. Les valeurs de paramètres utilisées ont modélisé une solution de communication suffisamment réactive pour que les fenêtres de basculement, même réduites, soient suffisantes.

La configuration des défaillances possibles d'IDS et de nœuds permet de limiter les évolutions possibles du modèle en fonction de la partie de la spécification EPL-HA que l'on souhaite vérifier pour une architecture donnée. Par contre, nous n'avons pas voulu imposer de scénario d'occurrence des défaillances autorisées. Nous ne voulions pas risquer d'omettre le ou les scénarios contredisant les propriétés attendues.

Sur l'automate UPPAAL modélisant le comportement d'un nœud, nous avons mis en place la possibilité de limiter les transitions vers l'état défaillant. Depuis l'extérieur, l'activité d'un nœud ne peut se constater que s'il envoie un message. La défaillance depuis n'importe quel état entre deux envois de message aura les mêmes conséquences qu'une défaillance provoquée seulement depuis l'état précédent l'envoi du second message.

Les transitions vers l'état défaillant de nœud ou d'IDS introduisent des indéterminismes qui favorisent l'explosion combinatoire. Ces basculements constituant le cœur des propriétés à vérifier, nous avons souligné que nous ne voulions pas les limiter. Par contre, un autre indéterminisme possible concerne le producteur de la trame ASnd dans la phase asynchrone. N'importe quel nœud est potentiellement producteur de cette trame. Nous avons introduit la possibilité de réduire les nœuds potentiellement producteurs de la trame ASnd à une liste à définir par l'utilisateur. Par exemple, pour modéliser une architecture en ligne, on ne pourra conserver que les nœuds les plus éloignés, c'est à dire entre lesquels le délai de communication sera le plus important.

Nous n'avons pas utilisé d'automates observateurs pour abstraire notre modélisation. Nous avons préféré placer des états de détection d'erreur à l'intérieur des automates UPPAAL :

- l'état AMN_TcycError pour un nœud,
- l'étatRedundantFrameMismatchIssue pour un LS,
- l'état FrameCollisionOrConfigurationIssue pour une IDS.

Nous utilisons également le contrôle des valeurs de variables du modèle, les variables booléennes `isAMN` et les variables entières `FramelossDetected` des instances de la classe Nœud.

3.6.3 Complexité du modèle

Malgré les abstractions mises en œuvre lors de l'élaboration du modèle, celui-ci conserve un certain niveau de complexité. Afin d'illustrer cette complexité, nous avons établi le tableau 3.6 suivant qui recense le nombre d'états par automate (ou sous-sytème) UPPAAL, le nombre de combinaisons de valeurs des variables privées de chaque automate UPPAAL ainsi que le nombre de combinaisons des valeurs des variables globales du système. Le tableau précise également le nombre d'horloges privées de chaque sous-système ainsi que le nombre d'horloges globales du système.

Pour avoir une idée du nombre d'états de contrôle possible après composition du modèle UPPAAL (même si toutes les combinaisons ne peuvent pas être réellement atteintes), il suffit de faire le produit :

$$NumberOfNodes \times NbEtats_{Noeud} \times NbCombinaisonsVariables_{Noeud} \times$$
$$NumberOfIDS \times NbEtats_{IDS} \times NbCombinaisonsVariables_{IDS} \times$$
$$NumberOfNodes \times NbEtats_{LS} \times NbCombinaisonsVariables_{LS} \times$$
$$NbCombinaisonsVariables_{Systeme}$$

	NbEtats	NbCombinaisonsVariables	NbHorloges
Sous-syst. nœud	14	2	2
Sous-syst. IDS	3	$2^{(NumberOfNodes+NumberOfIDS)+1}$	1
Sous-syst. LS	5	2	1
Système	–	$3 \times NumberOfNodes^3 \times (AMNFailuresMax + 1) \times (NodeFailuresMax - AMNFailuresMax + 1) \times 2^{NumberOfNodes \times NumberOfIDS + 2} \times (NumberOfNodes + NumberOfIDS)$	0

TABLE 3.6 – Complexité d'un modèle

Pour l'exemple de la figure 3.26, constitué de 4 nœuds et 10 IDS, le calcul donne $1,95 \cdot 10^{26}$ états de contrôles possibles. Nous allons voir dans le chapitre suivant que notre modélisation permet de limiter les combinaisons réellement atteignables et d'obtenir des résultats sur différents types d'architecture.

3.7 Conclusion sur le chapitre

Dans ce chapitre nous avons choisi une technique de vérification formelle en privilégiant l'exhaustivité de la vérification et la prise en compte des aspects temporels du système vérifié. Nous avons ensuite présenté la technique choisie, le model-checking temporel, ainsi qu'un outil associé, UPPAAL. Enfin, nous avons précisé la démarche et la modélisation obtenue.

L'objectif était de proposer une modélisation générique d'une architecture et des propriétés ne nécessitant que la modification d'un champ de configuration du modèle. C'est pourquoi nous avons recensé l'ensemble des paramètres constituant ce champ. Certains de ces paramètres décrivent le réseau physique sous la forme d'IDS. Nous avons proposé la modélisation par l'intermédiaire d'IDS également pour favoriser la généricité du modèle d'architecture.

Le chapitre suivant va nous permettre de mettre en oeuvre la modélisation proposée pour vérifier les propriétés sur une famille d'architecture ainsi que sur une architecture particulière. Les résultats obtenus nous permettront de valider la pertinence de l'extension EPL-HA sur la famille d'architecture et de confronter la modélisation au problème d'explosion combinatoire inhérent à la technique choisie.

Chapitre 4

Vérification formelle d'architectures

DANS ce chapitre, la modélisation générique est utilisée pour assurer la vérification des propriétés sur différents modèles d'architecture et de défaillance.
Les résultats obtenus sont la vérification des propriétés, mais aussi des indications de performance de la technique de vérification formelle choisie. Les architectures considérées ont pour objet l'évaluation d'EPL-HA d'une part à la modernisation de cellules existantes, d'autre part à l'intégration à une solution critique.

Sommaire

4.1 Vérification d'une famille d'architecture : description

Si le modèle élaboré est suffisamment générique pour décrire toute sorte d'architecture, la possibilité de vérifier les propriétés énoncées à la fin du chapitre précédent (section 3.5) n'est pas garantie.

Lors du choix d'une technique de vérification, nous avons vu que la vérification par model-checking s'accompagnait d'un phénomène d'explosition combinatoire. Ce phénomène a été mis en évidence dans la sous-section 3.6.3 avec l'évaluation de la compléxité du modèle. Plus l'architecture à vérifier est importante du point de vue du nombre de nœuds (donc du nombre de LS), plus le découpage des média en IDS est fin, plus le nombre d'états de contrôle potentiellement atteignables est élevé. La conséquence est qu'il existe des tailles de modèle et des degrés de possibilités de défaillance pour lesquels la quantité maximum de mémoire offerte par un calculateur sera atteinte. De plus, il existe une durée de calcul maximum qu'un concepteur d'architecture est disposé à accorder au moteur de model-checking pour que ce dernier lui fournisse une validation.

Parmi les abstractions qui ont été retenues lors de l'élaboration du modèle, le paramétrage des défaillances autorisées, par exemple, offre différentes options de vérification ou d'infirmation des propriétés. Deux raisons peuvent justifier de réduire les défaillances étudiées :

- rendre la vérification possible sur un modèle faisant intervenir plus de nœuds ou d'IDS que supportés lorsque les défaillances étudiées sont plus variées,
- réduire le temps de calcul nécessaire à la vérification lorsque l'on a moins d'exigence sur le domaine de validité des propriétés.

Nous nous proposons d'évaluer le champ de combinaisons des paramètres pour lesquels le modèle générique reste vérifiable. Nous allons quantifier ces combinaisons sur une famille d'architecture par l'intermédiaire d'une campagne de vérification. Les critères utilisés sont l'espace mémoire et le temps de calcul nécessaire à la vérification formelle. La limitation à une famille nous permet de donner des relations entre certains paramètres et limiter l'étendue de la campagne.

Les architectures considérées par la campagne sont des architectures en ligne telle que celle que nous avons utilisée dans l'exemple de la figure 3.11. Cette architecture correspond aux architectures mises en places par Alstom Power avec l'utilisation des bus de terrain. Elle ne présente pas une utilisation optimale d'un réseau de terrain sur Ethernet car elle nécessite d'introduire des équipements d'interconnexion avec chaque nouveau nœud.

Avec cette architecture, nous ne nous intéressons donc pas à la conception des cellules d'automatisme de futurs centres de production d'énergie. Nous nous intéressons au cas de la modernisation de cellules existantes pour lesquelles les emplacements des équipements et les chemins de câbles sont déjà plus ou moins imposés.

La figure 4.1 illustre le motif d'architecture qui va être répété en ligne ainsi que le diagramme d'objets correspondant.

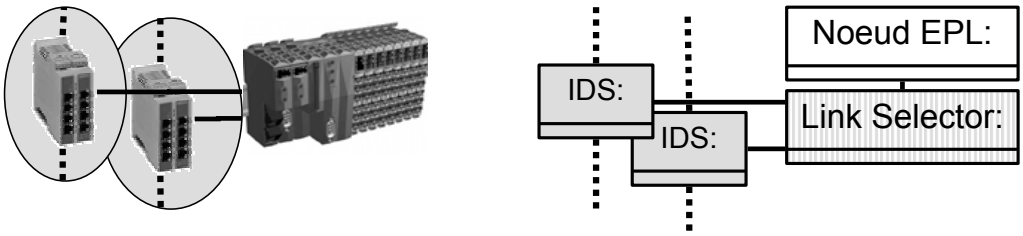


FIGURE 4.1 – Motif utilisé pour la vérification d'une architecture en ligne.

Avec cette architecture, chaque médium constitue une dorsale sur laquelle sont raccordés les nœuds par l'intermédiaire de concentrateurs dédiés. À partir du nombre de nœuds `NumberOfNodes`, on déduit directement :

- le nombre d'IDS `NumberOfIDS=2`×`NumberOfNodes`,
- l'attribution des identifiants des couples (nœuds ;LS) et des LS,
- les matrices de liaison entre les instances d'LS et d'IDS `LSPortsAssignement` et entre les différentes instances d'IDS `LinksBetweenIDS` décrivant la topologie,
- le programme de la phase isochrone `PollProg` qui prévoit la consultation de tous les nœuds dans l'ordre de leur identifiant.

Un seul type de nœud est considéré pour que tous les nœuds aient les mêmes paramètres temporels de description. Les valeurs choisies pour ces paramètres de description correspondent à des moyennes hautes des valeurs constatées sur différents équipements EPL. Nous conservons également des paramètres de configuration communs aux différentes expériences. Finalement, afin de limiter l'espace des états à parcourir, les producteurs potentiels de la trame ASnd sont les deux nœuds extrêmes de l'architecture en ligne, c'est-à-dire les nœuds de plus petit et de plus grand identifiant. Les états à partir desquels un nœud est autorisé à passer à l'état défaillant sont limités. Le tableau 4.1 restitue les valeurs des paramètres de description et de configuration des nœuds ainsi que les paramètres de réduction de l'espace des états que nous avons utilisés pour l'ensemble des expériences :

Paramètre de description	Valeur
`NodeJitter[i]`	3 000 ns
`tSoC2PReq[i]`	30 000 ns
`tPRes2PReq[i]`	30 000 ns
`tPRes2SoA[i]`	30 000 ns
`tSoA2ASnd[i]`	30 000 ns
`tPReq2PRes[i]`	6 000 ns
Paramètre de configuration	Valeur
`PResTimeout[i]`	55000 ns
`Tcyc`	1 000 000 ns
`PollProg`	0,2,3,...,n
Paramètre d'abstraction	Valeur
`AsndLimiterEnabled`	true
`ASndSenderIDLimiter`	0,n
`AllStatesFailureEnabled`	false

TABLE 4.1 – Paramètres de description et de configuration des nœuds

Toutes les IDS représentent la même structure type de médium avec les mêmes paramètres temporels. Les valeurs de ces paramètres correspondent à une IDS contenant un concentrateur et une longueur de câble limitée. La valeur du délai introduit est représentative de ce qui serait mesuré sur l'ensemble modélisé par l'IDS. Par contre, la valeur de la gigue introduite est surestimée bien que correspondant à l'unité de temps du modèle. Le tableau 4.2 suivant donne les valeurs choisies pour le délai et la gigue des différents IDS pour l'ensemble des expériences.

Paramètre de description	Valeur
`IDSLatency[i]`	1 000 ns
`IDSJitter[i]`	200 ns

TABLE 4.2 – Paramètres de description des IDS

Les résultats obtenus portent nécessairement sur la vérification de propriétés pour les différentes instances du modèle. Toutefois, nous nous intéressons également aux limites d'utilisation du modèle. Nous avons choisi l'étendue de la campagne de manière à relever la quantité de mémoire utilisée pour la vérification ainsi que l'évolution de cette utilisation au cours du temps. Le résultat de la vérification sur notre modélisation (et ses choix d'abstraction) nous ont permis de démontrer la qualité des principes de la solution EPL-HA vis-à-vis des propriétés attendues.

Sur l'ensemble des paramètres rappelés dans la figure 3.24, nous allons nous intéresser aux combinaisons de différentes valeurs des paramètres indiquant l'étendue de l'architecture (déduite de la valeur du paramètre `NumberOfNodes`) ainsi que les possibilités de défaillance autorisées (paramètres de défaillance des modèles de nœud et d'IDS). Le tableau 4.3 indique les noms des paramètres auxquels nous nous intéressons dans la campagne et leur plage de valeurs respective.

Paramètre de la campagne	Valeurs
`NumberOfNodes`	4, 5, 6, 7, 8, 9, 10
(`IDSFailuresEnabled`;`IDSFixingAllowed`)	(true ;true), (true ;false), (false ;false)
(`NodeFailureMax`;`AMNFailureMax`)	(0 ;0), (1 ;1), (2 ;2), (2 ;1)
`NodeFixingAllowed`	true, false

TABLE 4.3 – Paramètres de la campagne pour l'ensemble des vérifications

Dans la section 1.3 précisant le cahier des charges de la cellule d'automatismes, nous indiquions que celle-ci devait pouvoir accueillir jusque 30 nœuds. Pour la campagne, nous limitons à 10 le nombre possible de nœuds dans la cellule modélisée. Ce chiffre maximum correspond mieux aux premières applications d'Alstom utilisant EPL comme réseau de terrain. Nous limitons également à 2 le nombre possible de nœuds pouvant être défaillants en même temps. Au-delà de ce chiffre, nous considérons que trop de nœuds sont défaillants en même temps pour que le processus reste pilotable. L'origine de ces défaillances simultanées doit être trop sérieuse pour que la continuité du service de communication suffise à la compenser.

Chaque combinaison des valeurs des paramètres de réglage du tableau 4.3 a donné lieu à l'instanciation d'un modèle à partir du modèle générique, soit 147 modèles différents au total (si `(NodeFailureMax ;AMNFailureMax)=(0 ;0)`, `NodeFixingAllowed=true` donne les mêmes modèles que `NodeFixingAllowed=false`).

La vérification n'a été réalisée que sur un sous-ensemble de ces modèles à l'aide de l'exécutable `verifyta` accompagnant la version 4.0.6 d'Uppaal : par exemple, si un modèle n'était pas vérifiable pour `NumberOfNodes=n` le modèle ne se différentiant que par le paramètre `NumberOfNodes=n+1`, plus complexe, n'était pas vérifié.

Les résultats ont été regroupés de manière à contrôler la vérifiabilité de l'architecture pour un nombre croissant de nœuds :

- lorsque la redondance matérielle seule est sollicitée `(NodeFailureMax ;AMNFailureMax)=(0 ;0)` et `NodeFixingAllowed=false`,
- lorsque la redondance logicielle seule est sollicitée `(IDSFailuresEnabled ;IDSFixingAllowed)=(false ;false)`,
- lorsque les deux sont sollicitées à la suite ou de façon simultanée.

La vérification a été réalisée sur un calculateur doté d'un processeur Core2Duo E4600 (Le moteur de model-checking n'utilise qu'un seul cœur). Le système d'exploitation utilisé est la distribution Fedora 8 de GNU-LINUX.

4.2 Vérification d'une famille d'architecture : résultats

Les résultats donnés dans cette section nous permettent de mettre en évidence le travail d'abstraction décrit dans la section 3.6.2. Sur les 147 modèles de la campagne, la vérification a abouti (sans atteindre la limite de mémoire disponible) pour les 87 moins complexes. 101 exécutions de `verifyta` ont été nécessaires pour définir les combinaisons de valeurs à partir desquelles la mémoire disponible n'est plus suffisante.

Mais surtout, les résultats permettent à Alstom de confronter la spécification d'EPL-HA à un type d'architecture utilisé et maîtrisé en validant sur celui-ci les principes de redondance matérielle et logicielle retenus.

4.2.1 Vérification de la redondance matérielle

La redondance matérielle correspond à la redondance du médium et la gestion de celle-ci par la fonction LS. La vérification du principe de redondance matérielle a consisté à autoriser l'évolution vers un comportement de défaillance d'une instance de modèle d'IDS. La vérification des propriétés a été réalisée pour le sous-domaine suivant des plages de valeurs définies dans le tableau 4.3 :

Paramètre de la campagne	Valeurs (redondance matérielle seule)
`NumberOfNodes`	4, 5, 6, 7, 8, 9, 10
(`IDSFailuresEnabled`;`IDSFixingAllowed`)	(true ;true), (true ;false), (false ;false)
(`NodeFailureMax`;`AMNFailureMax`)	(0 ;0)
`NodeFixingAllowed`	false

Table 4.4 – Paramètres de la campagne pour la redondance matérielle seule

Les résultats sont illustrés dans les diagrammes de la figure 4.2.

Le premier diagramme indique la quantité de mémoire maximum utilisée pour la vérification des propriétés des différents modèles vérifiables. Lorsqu'un modèle n'a pas pu être vérifié, car il nécessitait un espace de mémoire trop important (plus de 3Go de mémoire), le volume correspondant n'est pas représenté.

Le second diagramme donne le temps de calcul qui a été nécessaire pour les différents modèles. Un cadre pointillé signifie que le temps correspond au temps de calcul jusqu'à dépassement de la quantité de mémoire disponible et arrêt de la vérification.

Le dernier diagramme illustre l'évolution du nombre d'états stockés en fonction de l'évolution des différents paramètres. Il illustre également le fait que les états atteints à partir d'une paramétrisation donnée peuvent être contenus dans l'espace des états atteints pour une paramétrisation offrant plus de possibilités. Ainsi, l'espace des états atteints par un modèle sans défaillance paramétrée est contenu dans les espaces des états atteints par des modèles incorporant plus de défaillances.

En l'absence de toute défaillance (volumes blancs), il y a peu d'évolutions différentes possibles. Le seul indéterminisme porte sur le producteur du ASnd qui ne peut être ici qu'un des deux nœuds de début de ligne. Par conséquent, la vérification des propriétés prend moins d'une minute jusque 10 nœuds.

En présence d'une défaillance, réparable ou non, les propriétés ont pu être vérifiées jusque 9 nœuds. Le temps de calcul nécessaire à découvrir qu'un modèle à 10 nœuds et une défaillance d'IDS met en évidence un inconvénient de la démarche. Un modèle d'architecture important en terme de nœuds et d'IDS avec des paramètres offrant beaucoup de possibilités d'évolution sera probablement non vérifiable. Par contre, la complexité du modèle ne fera pas atteindre plus vite la limite de mémoire disponible, au contraire.

Un des objectifs de cette campagne, à savoir la nécessité de pouvoir avoir une idée de la vérifiabilité d'un modèle à partir d'un nombre de nœuds et d'IDS se justifie ici. Un utilisateur ne pourra accepter d'attendre plusieurs jours ou semaines pour arriver à la conclusion que son architecture était trop importante pour être vérifiable.

Compte tenu de l'évolution de la mémoire nécessaire, il sera difficile d'introduire un autre type d'indéterminisme pour notre architecture avec plus de 8 nœuds. En effet, si les propriétés sont vérifiées à 9 nœuds la mémoire nécessaire atteint déjà 1 Go si une défaillance d'IDS non réparable est introduite. Néanmoins, ce nombre de nœud est homogène avec les premières architectures développées. De plus, il est amplement suffisant pour montrer qu'un réseau chargé respecte les propriétés attendues.

L'introduction d'une défaillance d'IDS non réparable dans un modèle non défaillant a un coût en mémoire, temps de calcul et nombre d'états stockés dont le taux varie avec le nombre de nœuds. Par contre, le taux d'augmentation au passage d'un défaut IDS non réparable à un défaut IDS réparable et relativement constant en fonction du nombre de nœuds. La déduction de la vérifiabilité d'un modèle avec défaut IDS réparable depuis les résultats sans réparabilité sont facilités. Enfin, on peut noter que les modèles vérifiables le sont en moins d'une journée. Une vérification de redondance matérielle seule prenant au moins un jour pourrait être considérée comme ayant peu de chances d'aboutir.

D'après ces résultats, le travail de modélisation et d'abstraction a permis d'atteindre des performances de vérification très satisfaisantes compte tenu de la complexité des modèles à vérifier.

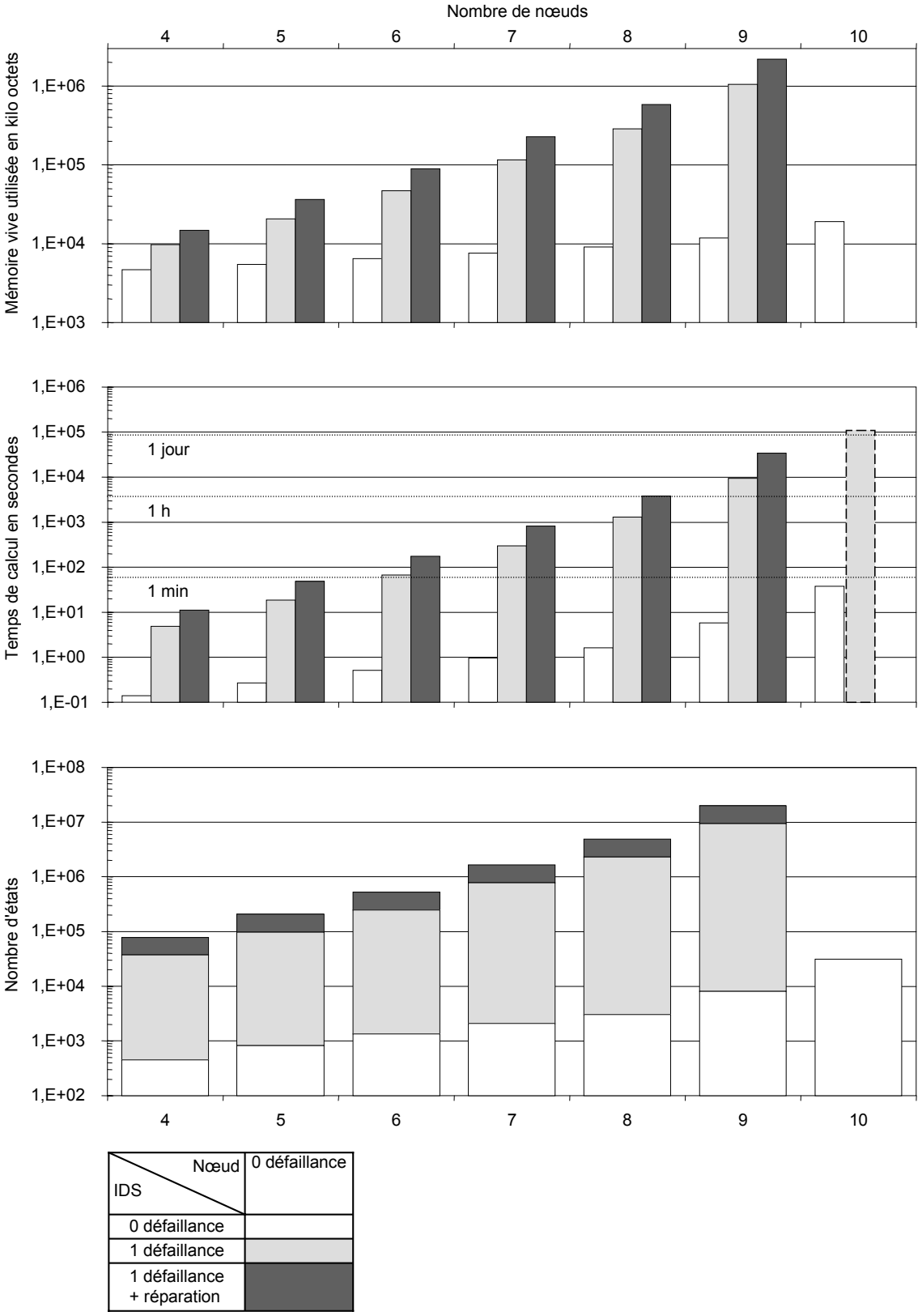


FIGURE 4.2 – Résultats sur sollicitation de la redondance matérielle

4.2.2 Vérification de la redondance logicielle

La redondance logicielle correspond à une redondance protocolaire et doit intervenir dès lors qu'une défaillance d'AMN a lieu. Pour vérifier sur le modèle que son intervention permet de conserver les propriétés formulées, nous autorisons la défaillance d'un ensemble d'instances de nœud. L'autorisation est réalisée par l'intermédiaire des paramètres `NodeFailureMax` et `AMNFailureMax`. La vérification des propriétés a été réalisée pour le sous-domaine suivant des plages de valeurs définies dans le tableau 4.3 :

Paramètre de la campagne	Valeurs (redondance logicielle seule)
`NumberOfNodes`	4, 5, 6, 7, 8, 9, 10
(`IDSFailuresEnabled`;`IDSFixingAllowed`)	(false ;false)
(`NodeFailureMax`;`AMNFailureMax`)	(0 ;0), (1 ;1), (2 ;2), (2 ;1)
`NodeFixingAllowed`	true, false

TABLE 4.5 – Paramètres de la campagne pour la redondance logicielle seule

Les couples définissant les possibilités de défaillance de nœud (1 ;1) et (2 ;2) autorisent respectivement la défaillance d'un nœud AMN et les défaillances consécutives de deux nœuds AMN. Le premier permet de vérifier la prise de relais de l'animation des échanges par un autre nœud dans le cas d'une défaillance. Le second permet de vérifier que la défaillance du nœud redondant n'occasionne pas la prise en défaut des propriétés attendues. Le couple (1 ;2) autorise la défaillance d'un nœud AMN et d'un nœud SMN. Ce couple permet de vérifier par exemple qu'en cas de défaillance simultanée du nœud AMN et de son nœud redondant le plus prioritaire, le relais est bien pris par un nœud tiers tout en conservant les propriétés.

Les résultats sont illustrés dans les diagrammes de la figure 4.3 . En l'absence de réparation, les propriétés sont vérifiables pour des architectures de 10 nœuds. Nous remarquons que le surcoût mémoire/temps est acceptable pour les trois paramétrisations de défaillance. De plus, le temps de vérification est inférieur à 10 minutes. La démarche de vérification formelle des propriétés est donc tout à fait pertinente pour le type d'architecture étudié lorsque les conséquences d'une réparation n'ont pas besoin d'être évaluées.

L'autorisation des réparations augmente l'espace des états atteignables. La vérification reste néanmoins possible à 10 nœuds pour une évolution autorisant un défaut d'AMN. L'évolution de la quantité de mémoire nécessaire pour un défaut AMN nous laisse supposer que l'introduction de tout autre indéterminisme sera difficile au delà de 8 nœuds.

La vérification d'évolutions autorisant deux défauts AMN ou un défaut AMN et un défaut SMN reste possible jusque 5 nœuds pour la campagne. L'introduction d'un défaut IDS en combinaison de ces dernières évolutions est également réalisable sur cette architecture dès lors que l'on réduit son exigence à 4 nœuds.

Contrairement aux résultats obtenus sur la redondance matérielle seule, la vérification de la redondance logicielle seule offre des exemples de calcul pour lesquels la vérification des propriétés a abouti même après un jour ou plus de calculs. En particulier, c'est le cas de l'évolution autorisant 2 défaillances de nœud AMN pour une architecture contenant 5 nœuds. Le calcul a duré près d'une semaine avant de conclure à la vérification des propriétés. En introduisant trop de complexité, la vérification s'avérerait trop longue pour vérifier différentes architectures à des fins de comparaison. Par contre, une vérification à 5 nœuds sur une architecture d'ores et déjà choisie permettrait de démontrer la qualité d'une offre.

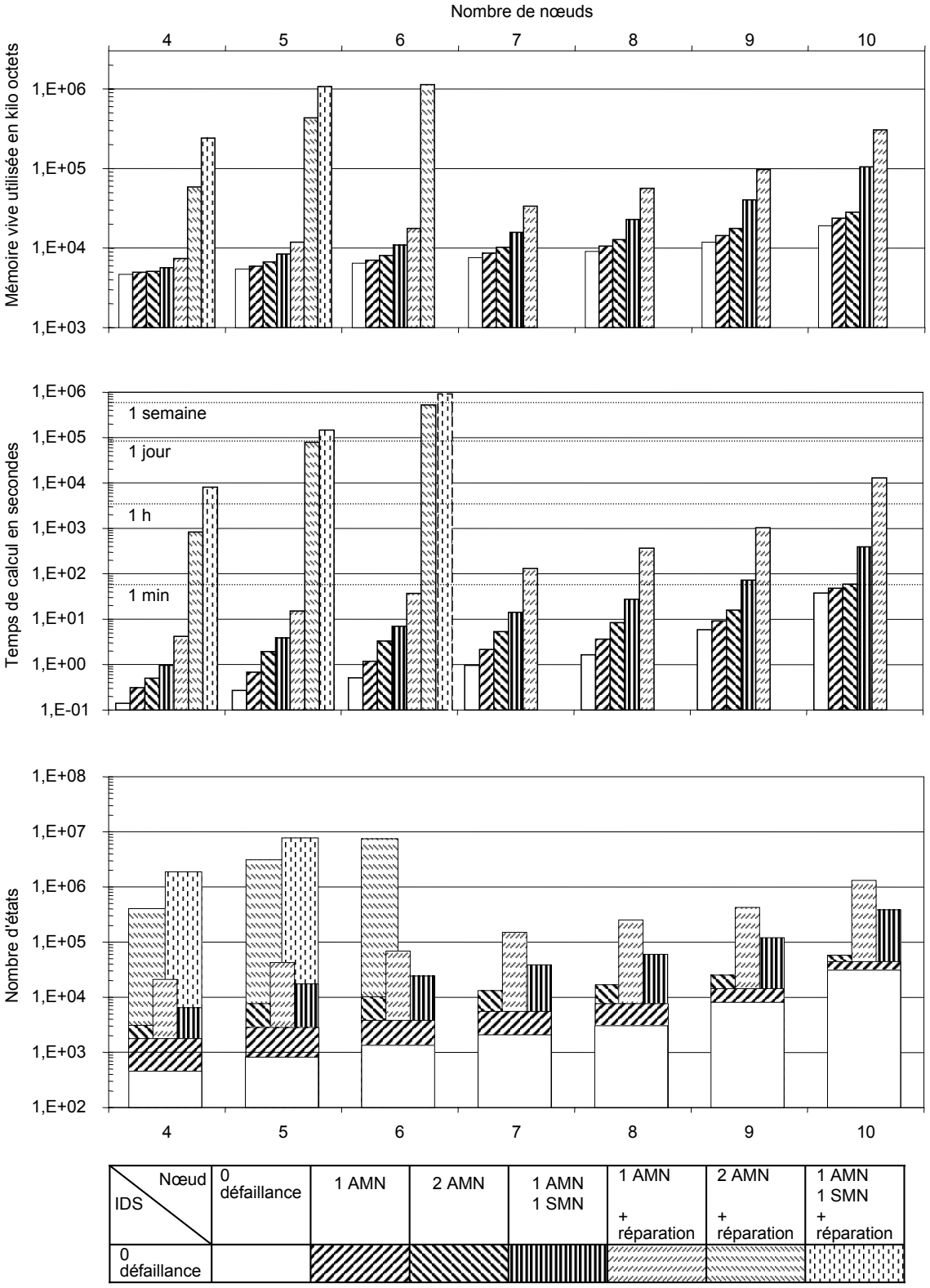


FIGURE 4.3 – Résultats sur sollicitation de la redondance logicielle

4.2.3 Redondances logicielle et matérielle simultanées

La vérification du principe de redondance logicielle a consisté à autoriser l'évolution sollicitant la combinaison des redondances matérielles et logicielles vérifiées dans les sous sections précédentes 4.2.1 et 4.2.2.

Compte tenu des résultats obtenus en séparant les deux types de redondances, les limites qui ont été estimées correspondent globalement aux résultats obtenus et illustrés en figure 4.4.

En l'absence de réparation de nœud, la vérification des propriétés est possible jusque 6 nœuds pour les trois évolutions de défaillance autorisées.

Même en ne considérant 4 nœuds, les valeurs de paramètres autorisant la défaillance simultanée de deux nœuds et leur réparabilité ne sont pas vérifiables en combinaison avec une défaillance d'IDS.

La vérification d'architectures autorisant défaillance d'IDS et défaillance réparable de nœud n'a été possible que pour l'évolution considérant une seule défaillance d'AMN. Jusque 5 nœuds, il est possible de vérifier les propriétés pour une évolution autorisant un défaut IDS et une défaillance de nœud en tant qu'AMN réparable.

Certainement en héritage des conclusions de la partie 4.2.1, tous les modèles vérifiables l'ont été en moins de 6 heures.

Cette série de vérifications permet de démontrer que tout enchaînement impliquant une défaillance de médium et sa réparation ainsi qu'une défaillance d'AMN et sa réparation se déroule en observant les propriétés formulées.

Par conséquent, nous avons réussi à démontrer que la spécification permet de répondre aux exigences fonctionnelles énoncées pour une cellule d'automatisation utilisant peu de nœuds pour ces enchaînements.

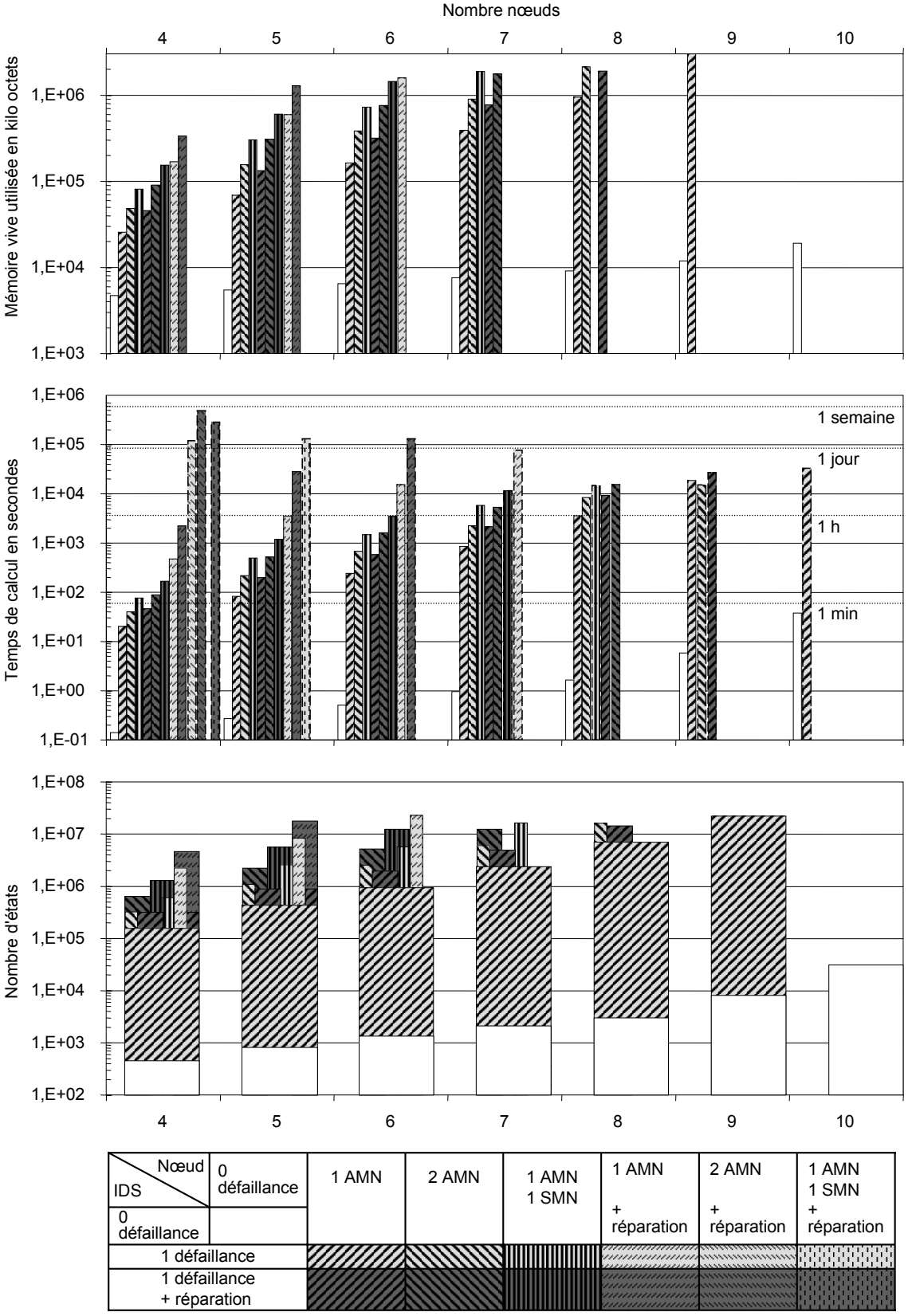


FIGURE 4.4 – Résultats sur sollicitation des redondances matérielle et logicielle

4.2.4 Conclusion

La vérification des propriétés a été effectuée pour une famille d'architecture en ligne. En fonction des exigences de possibilités de défaillance, le nombre de nœuds de l'architecture doit être limité pour que le modèle reste vérifiable en pratique sur un calculateur standard.

Du point de vue des applications de contrôle-commande de processus de production d'énergie, nous avons en partie validé la spécification de la solution EPL-HA. Nous avons vérifié les propriétés sur des architectures en ligne (par exemple pour la mise à jour de cellules en fonctionnement) sur des cellules contenant 5 nœuds :

- pour n'importe quel enchaînement d'états contenant une défaillance réparable de nœud et une défaillance réparable d'IDS,
- pour n'importe quel enchaînement d'états contenant deux défaillances réparables de nœud (avec au plus une défaillance de SMN), sans défaillance d'IDS.

En outre, aucun contre-exemple n'a été trouvé par le moteur de model-checking lors des vérifications n'ayant pas abouti.

Cette campagne d'essais a donc permis de valider la spécification EPL-HA pour 5 nœuds en ligne avec des valeurs réalistes des paramètres de description et de configuration des différents composants pour une seule sollicitation de la redondance logicielle et de la redondance matérielle. Si ce résultat témoigne la qualité de la spécification EPL-HA, il ne permet pas de généraliser le respect des propriétés pour toute architecture.

C'est pourquoi la section suivante s'intéresse à une architecture particulière ayant une topologie différente. Elle va également nous permettre de conclure sur la généralisation des limites de vérifiabilité d'une architecture (en nombre de nœuds et d'IDS) observées dans cette section.

En complément de ces résultats, nous avons vérifié que la configuration du modèle est susceptible de conduire à l'infirmation des propriétés attendues. Nous avons utilisé l'architecture à 6 nœuds, en l'absence de défaillance de nœud :

- lorsque aucune défaillance d'IDS n'est étudiée `(IDSFailuresEnabled ;IDSFixingAllowed)=(false ;false)`,
- lorsque une défaillance non réparable est étudiée `(IDSFailuresEnabled ;IDSFixingAllowed)=(true ;false)`.

Nous avons procédé par dichotomie pour définir la valeur du temps de cycle configuré `Tcyc` à partir de laquelle il y a dépassement (la propriété P3 n'est pas vérifiée cf. 3.5). Le tableau 4.6 suivant donne les résultats de cette étude complémentaire.

Propriété P3	`IDSFailuresEnabled=false` `IDSFixingAllowed=false`	`IDSFailuresEnabled=true` `IDSFixingAllowed=false`
$430\mu s \leq$`Tcyc`	vérifiée	vérifiée
$370\mu s \leq$`Tcyc`$\leq 420\mu s$	vérifiée	non vérifiée
`Tcyc`$\leq 360\mu s$	non vérifiée	non vérifiée

TABLE 4.6 – Vérification de la propriété P3 (3.5) pour une architecture à 6 nœuds

Il est possible de trouver une valeur limite de temps de cycle configuré pour laquelle l'ensemble des propriétés n'est pas vérifié. La sollicitation de la redondance matérielle augmente cette valeur limite.

4.3 Etude de cas

4.3.1 Description

Nous désirons appliquer la démarche de vérification des propriétés à une architecture différente de la topologie précédente. Le but de cette étude de cas est à la fois :

- d'étudier une architecture envisagée suite à l'adoption d'EPL. Nous ne situons plus la vérification dans un contexte de modernisation de cellules existantes, mais de conception d'architectures tirant parti d'EPL) ;
- de confirmer les conclusions extraites de la campagne d'essais ;
- de confronter les limites de vérification obtenues avec les résultats de la section 4.2. Il serait intéressant de pourvoir déterminer des limites de vérification indépendantes de l'architecture étudiée.

L'architecture à laquelle nous nous intéressons dans cette partie est représentée en figure 4.5. Elle correspond à l'architecture utilisée dans la sous-section 3.6.1 pour illustrer la méthode d'instanciation du modèle générique. Son étendue, impliquant des durées de propagation du signal importantes, pourrait prendre en défaut les mécanismes de redondance établis par EPL-HA.

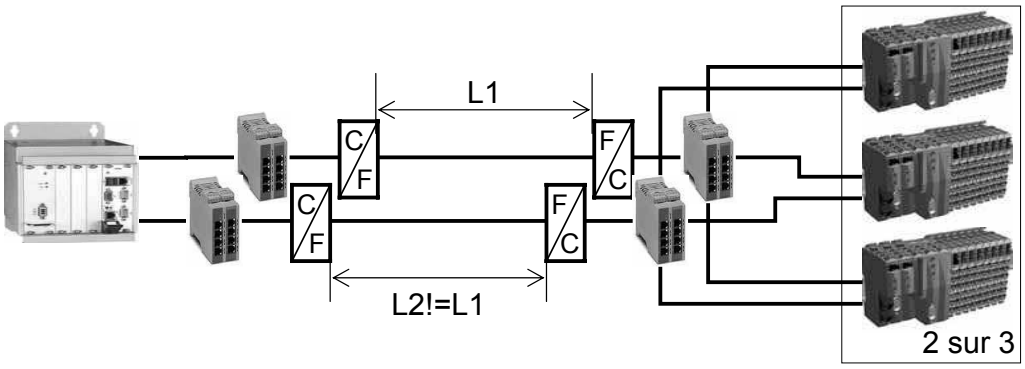


FIGURE 4.5 – Vue physique de l'architecture considérée

Cette architecture s'intéresse aux fonctions de contrôle des systèmes les plus critiques. Pour ces applications qui nécessitent une certification à un niveau de sûreté minimum, on met en place une redondance des équipements terminaux.

L'architecture évaluée utilise une topologie en étoile qui connecte 4 nœuds : un contrôleur de cellule et un système tripliqué. Les trois nœuds du système tripliqué assurent les mêmes fonctions d'acquisition, traitement et émission. Toutefois, les valeurs des sorties effectivement envoyées au procédé sont le résultat d'une comparaison et d'un vote entre les trois émissions.

La branche entre le contrôleur de cellule et le système tripliqué utilise une portion sur fibre optique, car le contrôleur de cellule est relativement éloigné de l'îlot système tripliqué. Nous considérons que la longueur de fibre est différente entre chaque médium (2000m et 2500m). En effet, les fibres auront des trajets différents dans la centrale pour assurer la redondance, c'est à dire pour qu'une erreur entraînant la coupure de l'un ne risque pas également d'entraîner la coupure de l'autre.

En cas d'isolement de l'îlot, nous voulons vérifier que les exigences fonctionnelles sont respectées (animation des échanges toujours assurée, pas de collision...) pour permettre la continuité du contrôle du système critique.

4.3.2 Modélisation

L'instanciation de l'architecture générique ayant été décrite dans la sous-section 3.6.1, le diagramme d'objet du modèle obtenu est donné par la figure 3.26.

Le modèle retenu est constitué de 4 instances des classes nœuds et LS. Ces instances utilisent des valeurs de paramètres temporels identiques à ceux utilisés dans la section 4.1.

Chaque médium est découpé en quatre instances d'IDS. Ce découpage nous permet d'empêcher la génération d'erreur due à un délai introduit par une IDS trop grand par rapport à la réactivité des nœuds (cf. 3.6.1). De plus, on obtient un nombre d'IDS légèrement supérieur à celui utilisé pour la campagne de vérification sur la famille d'architecture. Ceci nous permet de comparer les valeurs de mémoire, temps de calcul et nombre d'états nécessaires.

La modélisation de la différence de longueur est réalisée par l'intermédiaire du vecteur `IDSLatency` avec `IDSLatency={10,50,45,50,10,10,50,20,50,10}`. IDS(6) modélise une portion de câble plus longue qu'IDS(11).

Les paramètres de défaillance utilisés sont donnés dans le tableau suivant. Ce sont les mêmes que ceux utilisés pour la famille d'architecture.

Paramètre de l'étude de cas	Valeurs
(`NodeFailureMax`;`AMNFailureMax`)	(0;0), (1;1), (2;2), (2;1)
(`IDSFailuresEnabled`;`IDSFixingAllowed`)	(true;true), (true;false), (false;false)
`NodeFixingAllowed`	true, false

TABLE 4.7 – Paramètres de défaillance de l'étude de cas

À partir du tableau 4.7, 21 modèles ont été instanciés. Le champs de déclaration du modèle UPPAAL décrivant cette architecture (sans aucune défaillance autorisée) correspond au code ci-contre.

```
////////////////////////////////////////////////////////////////////////////////
// MODEL PARAMETERS WHICH NEED TO BE GIVEN //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________Architecture description_____________________________*/
const int NumberOfNodes=4;
typedef int [0,NumberOfNodes-1] idNode;
const int NumberOfIDS=10;

const int [0,2] LSPortsAssignement[NumberOfNodes][NumberOfIDS]={{1,0,0,0,0,2,0,0,0,0},
                                                                {0,0,0,0,1,0,0,0,0,2},
                                                                {0,0,0,0,1,0,0,0,0,2},
                                                                {0,0,0,0,1,0,0,0,0,2}};
const bool LinksBetweenIDS [NumberOfIDS][NumberOfIDS]={{0,1,0,0,0,0,0,0,0,0},
                                                       {1,0,1,0,0,0,0,0,0,0},
                                                       {0,1,0,1,0,0,0,0,0,0},
                                                       {0,0,1,0,1,0,0,0,0,0},
                                                       {0,0,0,1,0,0,0,0,0,0},
                                                       {0,0,0,0,0,0,2,0,0,0},
                                                       {0,0,0,0,0,2,0,2,0,0},
                                                       {0,0,0,0,0,0,2,0,2,0},
                                                       {0,0,0,0,0,0,0,2,0,2},
                                                       {0,0,0,0,0,0,0,0,2,0}};

const int [2,10000]IDSLatency[NumberOfIDS]={10,50,45,50,10,10,50,20,50,10};
const int [0,5000]IDSJitter[NumberOfIDS]={2,2,0,2,2,2,2,0,2,2};
const int [0,500] NodeJitter [NumberOfNodes] ={30,30,30,30};

const int [1,10000] tSoC2PReq[NumberOfNodes]={300,300,300,300};
const int [1,10000] tPRes2PReq[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPRes2PRes[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPRes2SoA[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tSoA2ASnd[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPReq2PRes[NumberOfNodes]={60,60,60,60};

const int [1,10000] PResTimeout[NumberOfNodes]={550,550,550,550};
const int [1,10000] ASndTimeout[NumberOfNodes]={800,800,800,800};
const int [4000,50000] Tcyc = 10000;
const int [NumberOfNodes-1,NumberOfNodes] PollProgSize=NumberOfNodes;
int [0,PollProgSize-1] PollProgStep;
const idNode PollProg [PollProgSize]={0,1,2,3};
/*_______________________________Failure Management description_____________________*/
const int [0, NumberOfNodes-1] NodeFailuresMax= 0;
const int [0, NodeFailuresMax] AMNFailuresMax= 0;
const int [0, NodeFailuresMax] SMNFailuresMax= NodeFailuresMax-AMNFailuresMax;
bool IDSFailuresEnabled= false;
const bool  NodeFixingAllowed= false && (NodeFailuresMax!=0);
const bool  IDSFixingAllowed= false;

typedef bool FailureEnabled;
const FailureEnabled SimplifiedSMNFailureEnabled= (SMNFailuresMax!=0);
const FailureEnabled AllStatesSMNFailureEnabled= SimplifiedSMNFailureEnabled && false;
const FailureEnabled SimplifiedAMNFailureEnabled= (AMNFailuresMax!=0);
const FailureEnabled AllStatesAMNFailureEnabled= SimplifiedAMNFailureEnabled && false;
/*_______________________________Manual non determinism reduction_____________________*/
const bool AsndLimiterEnabled=true;
const int [1,NumberOfNodes] NumberOfAsndEnabled=2;
const idNode ASndSenderIDLimiter [NumberOfAsndEnabled]={0,NumberOfNodes-1};
////////////////////////////////////////////////////////////////////////////////
// END OF MODEL PARAMETERS WHICH NEED TO BE GIVEN //
////////////////////////////////////////////////////////////////////////////////
```

4.3.3 Résultats

Sur les 21 modèles générés, 15 vérifications ont abouti pour 16 exécutions du moteur de model-checking. Les modèles ayant été vérifiées par ordre croissant de complexité, la vérification des 5 modèles restant n'aurait abouti qu'au dépassement de la quantité de mémoire possible. Les valeurs de mémoire vive, de temps de calcul et de nombre d'états atteints obtenues pour les différentes combinaisons de paramètres de défaillance sont données dans la figure 4.6.

Si l'on compare les résultats obtenus avec ceux d'une architecture de la section 4.2.3 à 5 nœuds, cette vérification des propriétés est plus exigeante en mémoire et temps de calcul. Une conséquence est que la vérification d'un modèle autorisant la réparation d'un nœud défaillant et la défaillance d'une IDS s'est révélée atteindre le maximum de mémoire disponible avant la fin du calcul.

Ces résultats montrent qu'on ne peut conclure sur la vérifiabilité d'un modèle d'architecture directement à partir de la quantité d'instances de composant qu'elle utilise. Pour un nombre équivalent d'IDS et de nœuds, le temps et la quantité de mémoire nécessaires à la vérification dépendent également de leur assemblage, c'est-à-dire de l'architecture étudiée. Une démarche incrémentale de vérification nous semble par conséquent adaptée à l'utilisation de cette modélisation. Il est préférable de commencer par vérifier un modèle offrant le minimum de possibilités de défaillance et étendre ses possibilités en fonction de l'objectif ainsi que du temps de calcul et de l'occupation mémoire observés à l'incrément précédent.

Le modèle nous a permis de vérifier les propriétés attendues pour toutes les combinaisons de défaillance sans réparation soumises.

D'après la modélisation, une cellule de contrôle commande constituée d'un contrôleur de cellule et d'un système tripliqué bénéficie de l'amélioration de disponibilité apportée par EPL-HA. En cas d'une coupure sur un médium et/ou de deux défaillances de nœud (dont le contrôleur de cellule), le système tripliqué est capable d'assurer les communications, donc son service offert, de manière autonome.

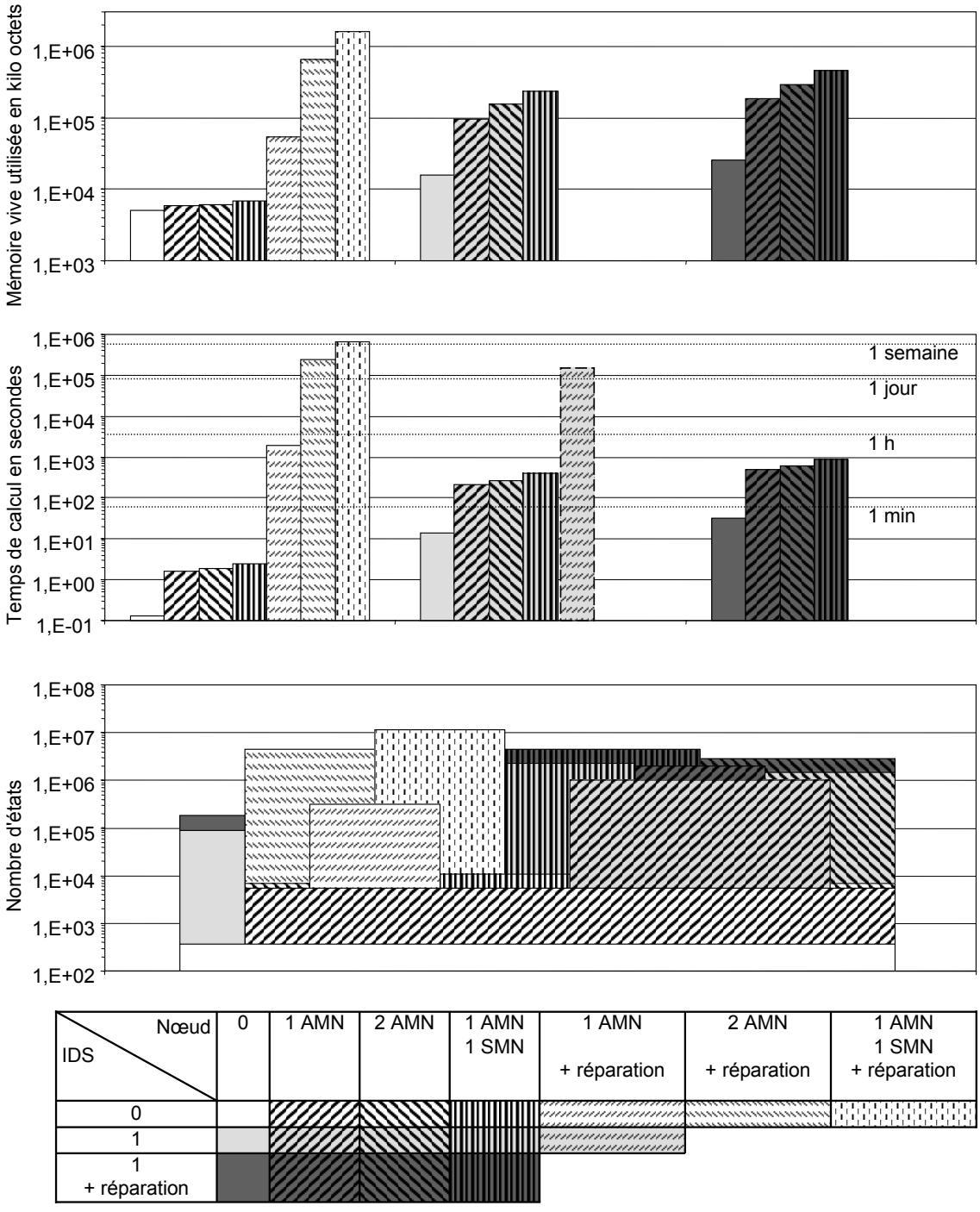


FIGURE 4.6 – Résultats de l'étude de cas.

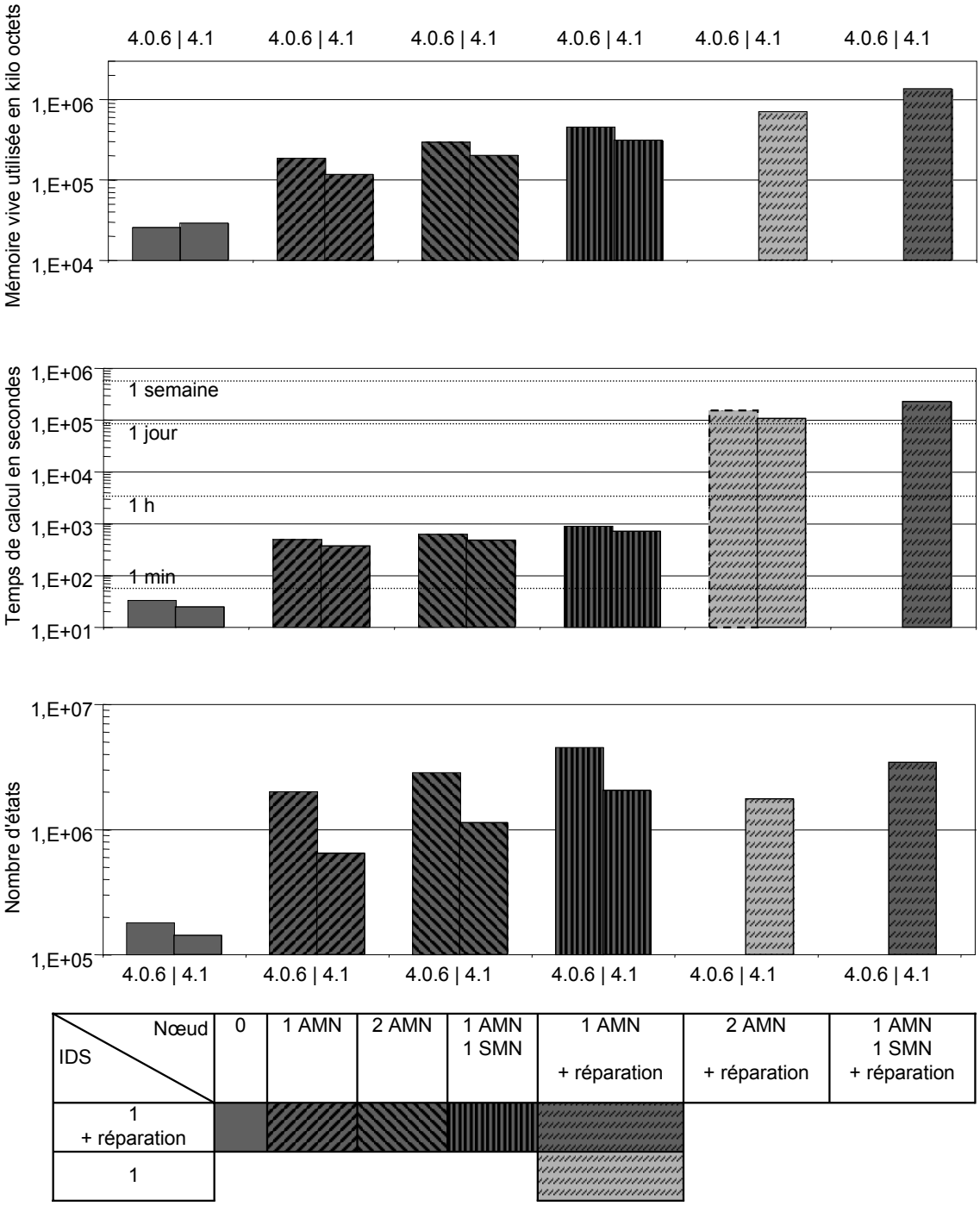


FIGURE 4.7 – Comparaison des résultats de l'étude de cas avec la version 4.1

Pendant la rédaction de ce document, l'équipe de développement d'Uppaal a mis à disposition une version récente et non stable permettant d'évaluer les améliorations à venir. Nous avons pu comparer les performances obtenues précédemment avec celles atteintes par cette version de développement (version 4.1 d'Uppaal). La figure 4.7 donne les résultats en version 4.0.6 et 4.1 pour quelques-unes des expériences précédentes.

Sur les modèles, on note que la nouvelle version de `verifyta` offre un gain d'environ 30% en temps de calcul et 50% en consommation mémoire sur les modèles vérifiables auparavant. Mais surtout, le gain se traduit par la possibilité d'étendre les possibilités de défaillance pour lesquelles les propriétés doivent être vérifiées.

La version de développement nous a permis de valider l'étude de la cellule dont l'architecture est représentée par la figure 4.6 pour tout enchaînement impliquant une défaillance d'IDS réparable et une défaillance d'AMN réparable.

4.4 Conclusion sur le chapitre

À l'issue de ce chapitre, nous pouvons affirmer que les propriétés attendues sont vérifiées sur deux applications de la modélisation générique. La première application est une famille d'architecture pour laquelle les nœuds sont raccordés à intervalle régulier sur une dorsale redondante. Cette architecture pourrait être utilisée dans le cas de modernisation de cellules existantes. La campagne de vérification qui a été menée a en outre montré quelles sont les limites de vérification de cette famille d'architecture. Compte tenu de la répétition du motif de composants, ces limites se traduisent par un nombre de nœuds pour un ensemble de possibilités de défaillance paramétré.

La seconde application s'intéressait au respect des exigences fonctionnelles par une architecture dédiée à la sûreté de fonctionnement. Nous avons montré que les limites de vérification ne résultaient pas seulement du nombre d'instances de nœuds et d'IDS utilisé par la modélisation, mais aussi de la topologie modélisée. Le modèle du cas d'étude considéré a néanmoins permis de conclure à la vérification des propriétés sur des possibilités de défaillance sans réparation.

Il sera à la charge de l'utilisateur de commencer à vérifier un modèle moins complexe et offrant le moins de possibilités de défaillances possible. En fonction des résultats obtenus, il pourra alors étoffer le modèle ou étendre les possibilités de défaillance.

En effet, il était prévisible que la mémoire disponible fixe la limite de vérification et que le temps de calcul varie en même temps que la taille du modèle. Par contre, nous avons constaté que le temps de calcul continue d'augmenter lorsque le modèle devient trop important pour être vérifiable. On aurait pu s'attendre à une explosion immédiate de l'espace d'états et une atteinte plus rapide du maximum de mémoire. Mais il est d'autant plus long de se rendre compte que le maximum mémoire est atteint que le modèle est complexe.

Nous avons pu évaluer l'activité et l'amélioration des moteurs de model-checking. La version de développement d'Uppaal testée nous a permis d'apprécier un gain notable en quantité de mémoire et en temps de calcul nécessaire à la vérification de modèles issus du cas d'étude.

Toutes versions d'Uppaal confondues, nous avons pu vérifier les propriétés recherchées pour les deux architectures étudiées pour des possibilités de défaillance relativement importantes. Les résultats obtenus sur ces deux applications permettent à Alstom de valider la spécification EPL-HA pour l'application à de petites architectures (avec un nombre faible de nœuds). Cette taille correspond aux premières applications proposées. Ces résultats permettent également à Alstom de démontrer sa maîtrise de cette nouvelle technologie respectivement pour de futures affaires et pour la certification de cellules dédiées à la sûreté.

Néanmoins, la conclusion la plus importante de ces campagnes d'essais est que pour tout enchaînement impliquant l'occurrence d'une défaillance d'AMN réparable et une défaillance d'IDS réparable la solution EPL-HA spécifiée n'a pas été prise en défaut.

Conclusion

Le pilotage de processus de production d'énergie impose de nombreuses exigences de sûreté de fonctionnement sur les technologies du système de contrôle-commande utilisé. En effet, les fonctions à remplir s'avèrent plus ou moins critiques pour la sécurité des biens et des personnes.

Le système de contrôle-commande ALSPA CONTROPLANT (Alstom Power) auquel nous nous sommes intéressés est un système distribué qui doit s'adapter à différents processus de production. Cette exigence d'adaptation concerne directement le support de communication de la cellule d'automatisation CONTROBLOC sur lequel nous nous sommes focalisés.

Ce support de communication, matérialisé par un réseau de terrain doit également répondre à l'évolution de l'exigence de réactivité. Pourtant, les processus n'ont pas radicalement évolué depuis les premiers systèmes distribués et les bus de terrains qui les supportent toujours. Néanmoins, l'amélioration de la réactivité répond à un besoin d'affinage du pilotage afin d'utiliser l'outil de production au maximum de ses capacités (sans l'endommager) et améliorer son rendement. L'augmentation du niveau de réactivité s'accompagne d'une augmentation de la quantité de données à échanger pour affiner le pilotage. Les niveaux demandés ne sont plus atteignables avec des solutions classiques de bus de terrain.

La quantité de données échangées fait partie des conditions de réactivité que nous avons retenues pour le réseau de terrain souhaité. Nous avons montré que l'exigence de réactivité que nous avons établie oriente les choix possibles vers une solution utilisant Ethernet (*Fast-Ethernet* ou *Ethernet-Gigabit*). En effet, portées par l'industrie, les performances d'Ethernet se sont développées beaucoup plus vite que les solutions de bus de terrain proposées spécifiquement pour l'industrie. Par conséquent, nombre des nouvelles solutions proposées sont regroupées sous l'appellation Ethernet Industriel.

La seconde exigence à laquelle doit répondre notre réseau de terrain est une exigence de déterminisme des échanges, car celui-ci participe au déterminisme du pilotage du processus. Peu de solutions Ethernet industrielles offrent déterminisme et niveau de réactivité voulu. Enfin, au moment de l'étude, aucune de ces solutions restantes ne satisfaisait la dernière exigence, la disponibilité. Celle-ci est nécessaire pour le pilotage de processus aux contraintes de sûreté de fonctionnement fortes. L'un des Ethernet Industriels offrant déterminisme et réactivité propose une extension permettant d'améliorer la disponibilité. Nos travaux ont eu pour objectif d'accompagner Alstom Power lors de l'intégration d'un bus de terrain à une solution Ethernet plus réactive que les solutions de bus de terrain maîtrisée jusqu'alors mais dépourvue des mécanismes de redondance satisfaisant à la criticité des processus à piloter. À l'occasion de ce saut technologique notre rôle a été de valider la spécification de l'extension EPL-HA par modélisation et vérification formelle par model-checking.

Ce document détaille la solution à modéliser, le déroulement des échanges sur un réseau Ethernet PowerLink ainsi que les mécanismes de redondance matérielle et de redondance logicielle spécifiés. L'utilisation de ces deux redondances doit permettre d'accéder à des propriétés de disponibilité par la solution. Nous avons énoncé ces propriétés. La redondance logicielle tire sa nécessité du modèle de communication Producteur - Consommateur - Distributeur adopté par EPL pour satisfaire à des exigences de réactivité et de déterminisme. Elle est donc étroitement liée au protocole d'échanges EPL et définit les règles de redondance passive du rôle de distributeur. La redondance matérielle spécifie une redondance active utilisant deux réseaux distincts et une fonction intermédiaire appelée Link Selector. Celle-ci s'intercale entre les fonctions remplies par un nœud EPL standard et le médium redondé. Le LS se charge de la sélection des trames redondantes entrantes et de la duplication de la trame sortante sur les deux réseaux.

Nous avons proposé une modélisation sous formes d'automates temporisés. La syntaxe utilisée correspond à l'atelier de vérification par model-checking temporel Uppaal mais pourrait être adaptée à un autre outil gérant le temps.

Le model-checking présente des risques d'explosion combinatoire empêchant de mener certaines vérifications à terme pour les modèles les plus complexes. Mais il offre surtout le moyen de parvenir à une validation exhaustive des propriétés. La validation par model-checking est d'ailleurs hautement recommandée pour la certification aux plus hauts niveaux de sûreté.

Nous avons montré que le comportement du système modélisé était fortement lié au temps. En fonction de la topologie de l'architecture choisie et de la réactivité intrinsèque des composants de l'architecture (nœuds ou composants d'interconnexion), les temps de communication et de déclenchement des mécanismes de redondance peuvent varier jusqu'à rendre inatteignable le niveau de réactivité souhaité.

La modélisation que nous avons présentée est suffisamment générique pour s'adapter aux différentes architectures possibles. Les modèles de deux architectures distinctes ne diffèrent que par la valeur des paramètres de description du modèle. La vérification d'une architecture ne nécessite pas la modification des modèles de comportement des différentes classes de composant.

Parmi les abstractions proposées pour s'affranchir des limites dues à l'explosion combinatoire, nous avons proposé et défini la modélisation d'un domaine de diffusion sous la forme de Sphère Isochrones de Diffusion. Une IDS modélise le délai nécessaire à un signal pour parcourir la partie du médium considérée. Nous avons également proposé une abstraction par repli du temps introduit par les trames. Cette proposition s'appuie sur le fait que la modélisation ne considère pas les réseaux commutés.

Il résulte que la méthode proposée d'utilisation du modèle se limite à donner l'ordre de renseignement des différents paramètres ainsi que les limites de découpage des domaines de diffusion en IDS. Aucune compétence en modélisation par automates temporisés ou vérification par model-checking n'est nécessaire pour exécuter la validation d'une architecture spécifiée.

Dans le dernier chapitre, nous avons validé les propriétés attendues sur deux types d'architectures.

La première est inspirée des architectures traditionnelles (en ligne) des bus de terrain et a permis de valider l'utilisation d'EPL-HA pour la modernisation de cellules d'automatisation pour des possibilités de défaillance relativement étendues. Il a été possible de démontrer en moins d'une demi-journée que, jusqu'à 5 nœuds en ligne, les propriétés attendues sont respectées pour toute combinaison :

- d'une défaillance réparable du nœud AMN,
- d'une défaillance réparable d'une partie d'un médium (i.e. d'une IDS). Nous avons ainsi quantifié le phénomène d'explosion combinatoire inhérent au model-checking pour cette famille d'architecture et montré la pertinence des abstractions choisies.

La seconde architecture s'intéresse à un cas d'étude particulièrement critique, susceptible d'être audité pour une certification SIL. Il s'agit d'un système *2 sur 3*, relié au réseau de supervision par l'intermédiaire du contrôleur de cellule et devant assurer des fonctions de pilotage en cas d'isolement. Les propriétés ont été vérifiées en moins de vingt minutes pour toute combinaison :

- d'une défaillance non réparable du nœud AMN,
- d'une défaillance réparable d'une IDS.

Cette seconde architecture fait l'objet d'une seconde série de vérification (des mêmes modèles) avec une version de développement d'Uppaal permettant d'augmenter la dimension des défaillances possibles. Notre modélisation bénéficie par conséquent des progrès réalisés par les outils de vérifications.

À la vue des performances atteintes et des améliorations de performance envisageables des outils de model-checking, la première perspective de ces travaux est d'étendre la modélisation par IDS. L'introduction d'une politique de mise en file d'attente à la classe IDS permettrait à celle-ci de ne plus être réservée au découpage de domaines de diffusion mais de pouvoir modéliser des réseaux commutés.

Cette extension des possibilités de modélisation de la classe IDS nécessiterait d'abandonner l'abstraction par repli du temps introduit par les trames et d'augmenter la complexité introduite par l'IDS. Toutefois, ceci permettrait également d'élargir le rayon des sphères (limité actuellement pas la réactivité des nœuds), donc de réduire le nombre éventuel d'instances d'IDS d'un modèle.

Une abstraction qui n'a pas été exploitée par notre modélisation est l'utilisation d'automates dits observateurs. Le but serait de gagner en temps de calcul et en quantité de mémoire nécessaire pour étendre la taille des modèles vérifiables.

Ainsi, des automates observateurs pourraient être utilisés pour provoquer l'occurrence des défaillances et observer leurs conséquences sur un nombre limité de cycles d'échanges EPL. On réduirait ainsi l'indéterminisme laissé par la modélisation pour l'apparition défaillances. Il suffirait alors que les propriétés soient vérifiées sur ce nombre réduit de cycles.

Par extension, les automates observateurs pourraient segmenter la vie de la cellule couverte actuellement par le modèle en différentes périodes de vie. On exécuterait alors une vérification par période de vie en partant de situations de contrôle différentes plutôt qu'une seule vérification sur l'ensemble de la vie de la cellule. Pour des exécutables 32bits n'utilisant qu'une seule unité de calcul tels que `verifyta`, ceci permettrait de profiter des possibilités de calcul parallèle et de la quantité de mémoire exploitable des calculateurs récents.

Une dernière approche que nous n'avons pas développée mais qui aurait beaucoup d'intérêt pour la conception d'architecture est d'utiliser le model-checking en tant qu'assistance au dimensionnement. Avec le modèle actuel, nous soumettons les valeurs des paramètres au modèle pour obtenir la validation ou non des propriétés. À l'instar de [68], il serait possible d'utiliser le model-checking pour déterminer, par exemple, le domaine de validité de paramètres de configuration des nœuds à partir de la description de l'architecture réseau. Il serait ainsi également possible de déterminer les domaines de rayons maximums des IDS pour lesquels les propriétés sont respectées et déduire les étendues limites d'architectures envisagées. Cette dernière approche pourrait nécessiter d'abandonner Uppaal et sa syntaxe et traduire la modélisation pour un autre outil de model-checking.

Modèle Uppaal complet d'une architecture

Cette annexe contient le modèle Uppaal complet d'une architecture en ligne telle que celle vérifiée dans la campagne de la section 4.1 et contenant 5 nœuds. Une vue de de cette architecture et du diagramme d'objet que nous avons retenu est donné par la figure 8.

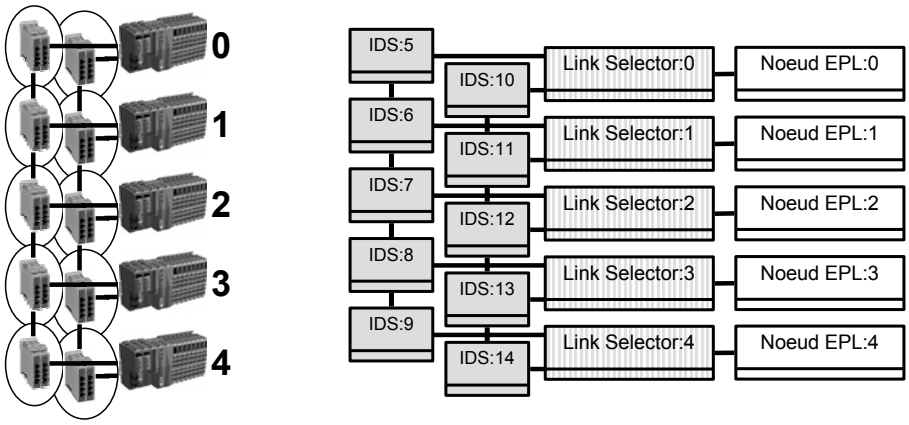
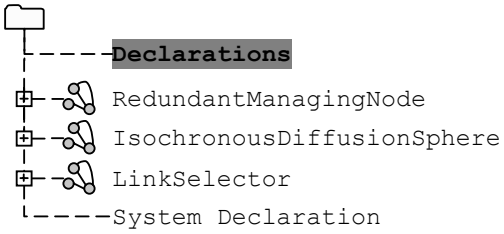


Figure 8 – Architecture modélisée et diagramme d'objet correspondant

Tous les paramètres détaillant les particularités de cette architecture sont regroupés sur les deux premières pages de code. Les paramètres de défaillance autorisent 1 défaillance réparable d'AMN et une défaillance réparable d'IDS.

Chaque champs de déclaration à été découpé de manière a présenter le code et les commentaires face à face, respectivement sur les pages paires et impaires.

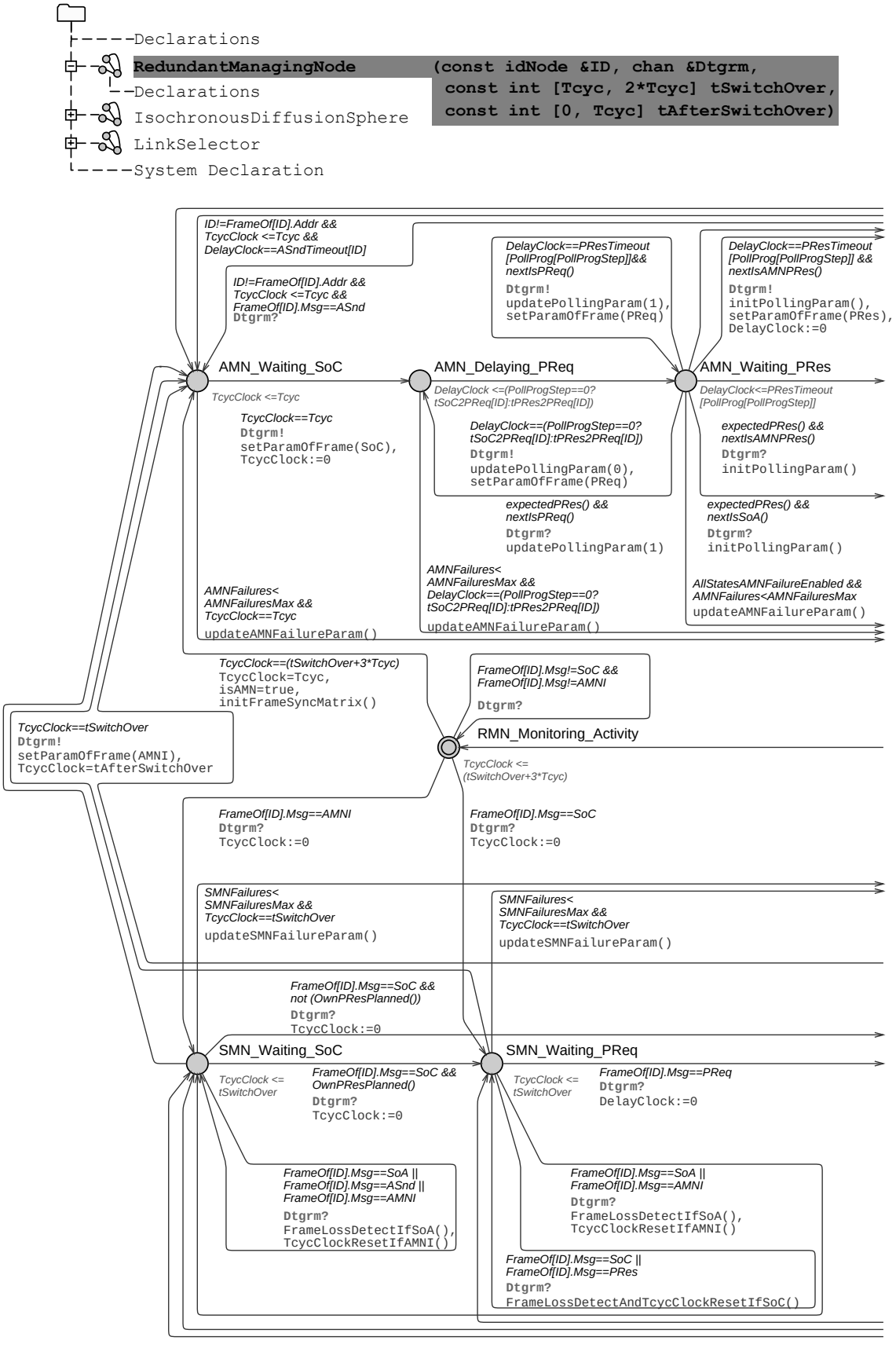


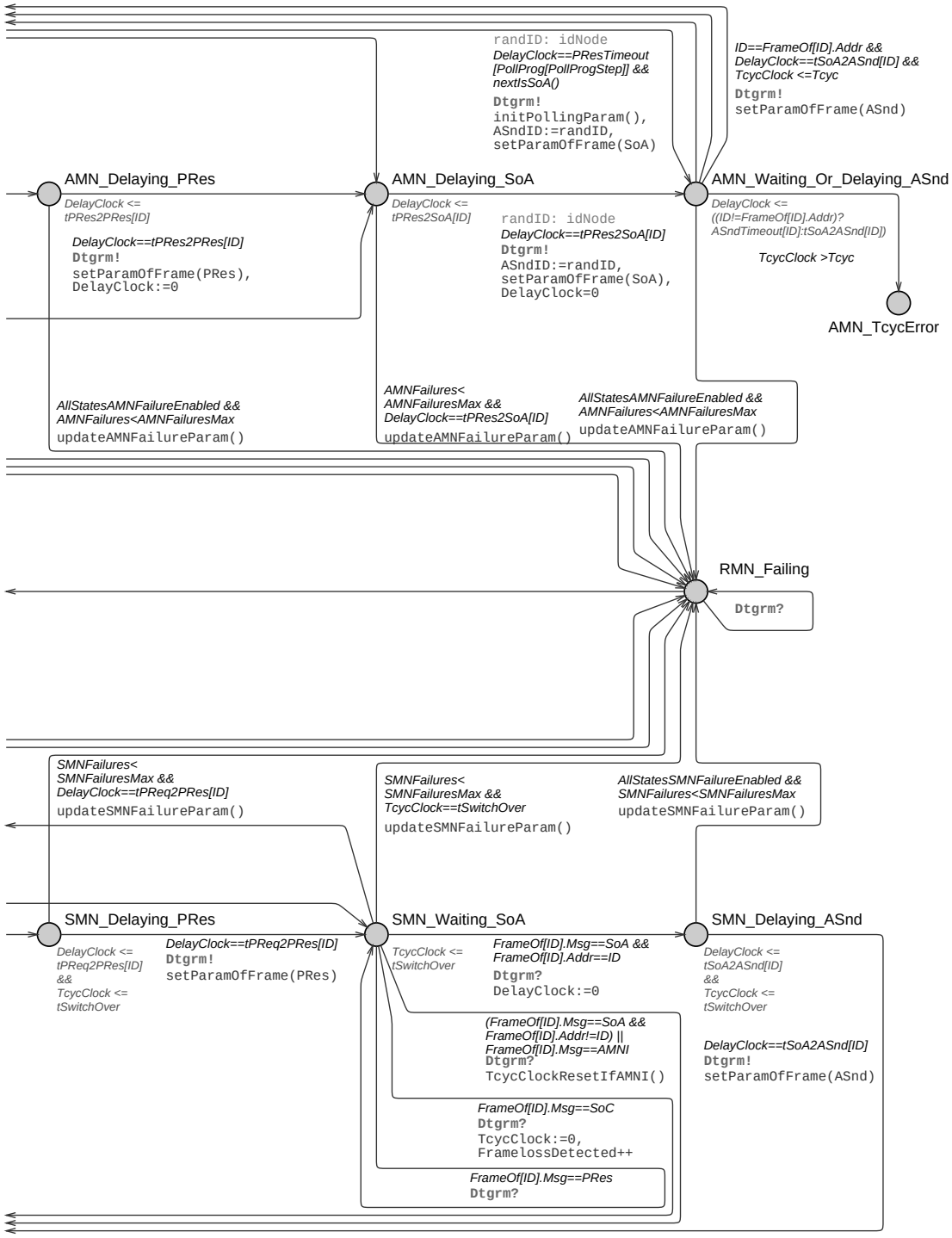

```
////////////////////////////////////////////////////////////////////////////////
//                   MODEL PARAMETERS WHICH NEED TO BE GIVEN CODE                //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________Architecture description CODE_____________________*/
const int NumberOfNodes=5;
typedef int [0,NumberOfNodes-1] idNode;
const int NumberOfIDS=10;

const int [0,2] NodePortsAssignement[NumberOfNodes][NumberOfIDS]=
{
{1,0,0,0,0,2,0,0,0,0},
{1,0,0,0,0,2,0,0,0,0},
{0,0,0,0,1,0,0,0,0,2},
{0,0,0,0,1,0,0,0,0,2},
{0,0,0,0,1,0,0,0,0,2}
};
const bool LinksBetweenIDS [NumberOfIDS][NumberOfIDS]=
{
{0,1,0,0,0,0,0,0,0,0},
{1,0,1,0,0,0,0,0,0,0},
{0,1,0,1,0,0,0,0,0,0},
{0,0,1,0,1,0,0,0,0,0},
{0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,2,0,0,0},
{0,0,0,0,0,2,0,2,0,0},
{0,0,0,0,0,0,2,0,2,0},
{0,0,0,0,0,0,0,2,0,2},
{0,0,0,0,0,0,0,0,2,0}
};

const int [2,10000]IDSLatency[NumberOfIDS]={10,50,20,50,10,10,50,20,50,10};
const int [0,5000]IDSJitter[NumberOfIDS]={2,2,0,2,2,2,2,0,2,2};
const int [0,500] NodeJitter [NumberOfNodes] ={30,30,30,30,30};

const int [1,10000] tSoC2PReq[NumberOfNodes]={300,300,300,300,300};
const int [1,10000] tPRes2PReq[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPRes2PRes[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPRes2SoA[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tSoA2ASnd[NumberOfNodes]=tSoC2PReq;
const int [1,10000] tPReq2PRes[NumberOfNodes]={60,60,60,60,60};

const int [1,10000] PResTimeout[NumberOfNodes]={550,550,550,550,550};
const int [1,10000] ASndTimeout[NumberOfNodes]={800,800,800,800,800};
const int [4000,50000] Tcyc = 10000;
const int [NumberOfNodes-1,NumberOfNodes] PollProgSize=NumberOfNodes;
int [0,PollProgSize-1] PollProgStep;
const idNode PollProg [PollProgSize]={0,1,2,3,4};
```

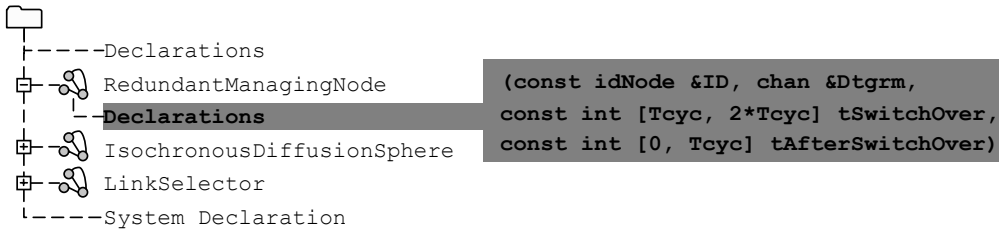
```
////////////////////////////////////////////////////////////////////////////////
//                 MODEL PARAMETERS WHICH NEED TO BE GIVEN COMMENTS                //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________Architecture description_____________________________*/
// UPDATE the number of Nodes(with LS) on the network
// node ID type building
// UPDATE the number of IDS on the network
// Topology of the network:
// * UPDATE "LS+Node" groups'ports (lines) to IDS (columns) it is syncing with
// (Both ports cannot listen to the same IDS)




// * UPDATE IDS (lines) to IDS (columns) it is syncing with
// ("1" and "2" values both mean "true" and are just for user)









// IDS and nodes description parameters:
//  * forwarding mean latency delays (IDS r,..., IDS s)x100ns
// * forwarding jitter (IDS r,..., IDS s)x100ns
// * Node's jitter when sending frame (node 1,..., node n) x 100ns
// Nodes description parameters, latency (node 1,., n) x 100ns between frames
// * UPDATE node required time between SoC and PReq
// * UPDATE RMN necessary time between PReq reception and PRes emission
// * UPDATE AMN necessary time between PRes reception and PReq emission
// * UPDATE AMN necessary time between PRes reception and PRes emission
// * UPDATE AMN necessary time between PRes reception and SoA emission
// * UPDATE node necessary time between SoA and Asnd emission
// Nodes configuration parameters
// * UPDATE Allowed delay (node 1,., n) x 100ns between PReq and suitable PRes
// * UPDATE Allowed delay (node 1,., n) x 100ns between SoA and ASnd
// * UPDATE Configured EPL cycle time x 100ns
//  -UPDATE Number of PRes in the cycle. 1 PRes per SMN (-1/+0)
// -DO NOT CHANGE variable storing the step achieved in the program
// -UPDATE the order Nodes are polled
```

```
/*_______________________________Failure Management description CODE_________________*/
const int [0, NumberOfNodes-1] NodeFailuresMax= 1;
const int [0, NodeFailuresMax] AMNFailuresMax= 1;
const int [0, NodeFailuresMax] SMNFailuresMax= NodeFailuresMax-AMNFailuresMax;
bool IDSFailuresEnabled= true;
const bool NodeFixingAllowed= true && (NodeFailuresMax!=0);
const bool IDSFixingAllowed= true;

typedef bool FailureEnabled;
const FailureEnabled SimplifiedSMNFailureEnabled= (SMNFailuresMax!=0);
const FailureEnabled AllStatesSMNFailureEnabled= SimplifiedSMNFailureEnabled && false;
const FailureEnabled SimplifiedAMNFailureEnabled= (AMNFailuresMax!=0);
const FailureEnabled AllStatesAMNFailureEnabled= SimplifiedAMNFailureEnabled && false;

/*_______________________________Manual non determinism reduction CODE_____________*/
const bool AsndLimiterEnabled=true;
const int [1,NumberOfNodes] NumberOfAsndEnabled=2;
const idNode ASndSenderIDLimiter [NumberOfAsndEnabled]={0,NumberOfNodes-1};
////////////////////////////////////////////////////////////////////////////////////
//                    END OF MODEL PARAMETERS WHICH NEED TO BE GIVEN CODE          //
////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////
//                    PARAMETERS MANAGED BY THE MODEL CODE                         //
////////////////////////////////////////////////////////////////////////////////////
/*_______________________________Failure Management CODE_____________________________*/

int [0, AMNFailuresMax] AMNFailures= 0;
int [0, SMNFailuresMax] SMNFailures= 0;
/*_______________________________Nodes and IDS identifier CODE_________________________*/
const int NumberOfNodes8IDS=NumberOfNodes+NumberOfIDS;

idNode ASndID;
typedef int [NumberOfNodes,NumberOfNodes8IDS-1] idIDS;
typedef int [0,NumberOfNodes8IDS-1] idNode8IDS;
/*_______________________________EPL messages CODE___________________________________*/
const int MessageIDMaxi=5;
typedef int [0,MessageIDMaxi] idMsg;
const idMsg SoC =0;
const idMsg PReq=1;
const idMsg PRes=2;
const idMsg SoA =3;
const idMsg ASnd=4;
const idMsg AMNI=5
/*_______________________________Event Synchronisations CODE__________________________*/
broadcast chan FrameSync[NumberOfNodes8IDS];
chan DatagramSync[NumberOfNodes];
bool ListensToIDS[NumberOfNodes8IDS][NumberOfIDS];
/*_______________________________Data linked to a frame event CODE____________________*/
typedef struct {idMsg Msg;idNode Addr;} FrameDefinition;
FrameDefinition FrameOf [NumberOfNodes8IDS];
/*_______________________________MN redundancy CODE___________________________________*/
const int All_MNSwitchOverCycleDivider=NumberOfNodes+(NumberOfNodes+1)/(NumberOfNodes);
const int SliceTime= (Tcyc/All_MNSwitchOverCycleDivider);
////////////////////////////////////////////////////////////////////////////////////
//                    END OF PARAMETERS MANAGED BY THE MODEL CODE                  //
////////////////////////////////////////////////////////////////////////////////////
```

```
// Allowed failure behaviour:
// * UPDATE Maximum number of nodes failling at the same time
// * UPDATE Maximum number of nodes failling at the same time from AMN state
// * DO NOT CHANGE Maximum number of nodes failling at the same time while SMN state
// * UPDATE IDS failure enabled
// * UPDATE Property of a node to heal or be fixed after a failure
// * UPDATE Property of an IDS to heal or be fixed after a failure
//Failures enabling variables:
// * DO NOT CHANGE Declaration of variable type
// * DO NOT CHANGE Simplified failures for SMN
// * From-every-state failures for SMN
// * DO NOT CHANGE Simplified failures for AMN
// * UPDATE From-every-state failures for AMN

// State space reduction:
// * UPDATE reduction of possible ASnd producers enabled
// * UPDATE number of ASnd producers allowed
// * UPDATE list of nodes'ID allowed to produce ASnd
////////////////////////////////////////////////////////////////////////////////
//                  END OF MODEL PARAMETERS WHICH NEED TO BE GIVEN COMMENTS        //
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//                   PARAMETERS MANAGED BY THE MODEL COMMENTS                    //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________Failure Management COMMENTS_____________________*/
//Failures counters:
// * AMN Failures
// * SMN Failures
/*_______________________________Nodes and IDS identifier COMMENTS________________*/
// total number of considered group of devices
// (a Node share its ID with its LS)
// ASnd sender identification
// IDS ID type building
// group of devices ID type building
/*_______________________________EPL messages COMMENTS____________________________*/
// Number of message types
// building of Message ID type
// EPL Message ID for SoC
// EPL Message ID for PReq
// EPL Message ID for PRes
// EPL Message ID for SoA
// EPL Message ID for ASnd
// EPL Message ID for AMNI
/*_______________________________Event Synchronisations COMMENTS__________________*/
// Synchronisation between Nodes and IDS, for frame emission or transmition
// Synchronisation between nodes and their Link Selector.
// Matrix storing currently allowed Frame event synchronisation between IDS and nodes
/*_______________________________Data linked to a frame event COMMENTS____________*/
// definition of the structure of data defined as a frame
// Data stored by each Node and IDS
/*_______________________________MN redundancy COMMENTS___________________________*/
// Cycle slicing number (which can be changed)
// Slice time value x 100ns (not to be changed)
////////////////////////////////////////////////////////////////////////////////
//                END OF PARAMETERS MANAGED BY THE MODEL COMMENTS                //
////////////////////////////////////////////////////////////////////////////////
```


Declarations
RedundantManagingNode (const idNode &ID, chan &Dtgrm, const int [Tcyc, 2*Tcyc] tSwitchOver, const int [0, Tcyc] tAfterSwitchOver)
Declarations
IsochronousDiffusionSphere
LinkSelector
System Declaration
AMN_Waiting_SoC
AMN_Delaying_PReq
AMN_Waiting_PRes
SMN_Waiting_SoC
SMN_Waiting_PReq
RMN_Monitoring_Activity
ID!=FrameOf[ID].Addr && TcycClock <=Tcyc && DelayClock==ASndTimeout[ID]
ID!=FrameOf[ID].Addr && TcycClock <=Tcyc && FrameOf[ID].Msg==ASnd
Dtgrm?
DelayClock==PResTimeout [PollProg[PollProgStep]]&& nextIsPReq()
Dtgrm! updatePollingParam(1), setParamOfFrame(PReq)
DelayClock==PResTimeout [PollProg[PollProgStep]] && nextIsAMNPRes()
Dtgrm! initPollingParam(), setParamOfFrame(PRes), DelayClock:=0
TcycClock <=Tcyc
TcycClock==Tcyc
Dtgrm! setParamOfFrame(SoC), TcycClock:=0
DelayClock <=(PollProgStep==0? tSoC2PReq[ID]:tPRes2PReq[ID])
DelayClock==(PollProgStep==0? tSoC2PReq[ID]:tPRes2PReq[ID])
Dtgrm! updatePollingParam(0), setParamOfFrame(PReq)
DelayClock<=PResTimeout [PollProg[PollProgStep]]
expectedPRes() && nextIsAMNPRes()
Dtgrm? initPollingParam()
expectedPRes() && nextIsPReq()
Dtgrm? updatePollingParam(1)
expectedPRes() && nextIsSoA()
Dtgrm? initPollingParam()
AMNFailures< AMNFailuresMax && TcycClock==Tcyc
updateAMNFailureParam()
AMNFailures< AMNFailuresMax && DelayClock==(PollProgStep==0? tSoC2PReq[ID]:tPRes2PReq[ID])
updateAMNFailureParam()
AllStatesAMNFailureEnabled && AMNFailures<AMNFailuresMax
updateAMNFailureParam()
TcycClock==(tSwitchOver+3*Tcyc)
TcycClock=Tcyc, isAMN=true, initFrameSyncMatrix()
FrameOf[ID].Msg!=SoC && FrameOf[ID].Msg!=AMNI
Dtgrm?
TcycClock==tSwitchOver
Dtgrm! setParamOfFrame(AMNI), TcycClock=tAfterSwitchOver
TcycClock <= (tSwitchOver+3*Tcyc)
FrameOf[ID].Msg==AMNI
Dtgrm? TcycClock:=0
FrameOf[ID].Msg==SoC
Dtgrm? TcycClock:=0
SMNFailures< SMNFailuresMax && TcycClock==tSwitchOver
updateSMNFailureParam()
SMNFailures< SMNFailuresMax && TcycClock==tSwitchOver
updateSMNFailureParam()
FrameOf[ID].Msg==SoC && not (OwnPResPlanned())
Dtgrm? TcycClock:=0
TcycClock <= tSwitchOver
FrameOf[ID].Msg==SoC && OwnPResPlanned()
Dtgrm? TcycClock:=0
TcycClock <= tSwitchOver
FrameOf[ID].Msg==PReq
Dtgrm? DelayClock:=0
FrameOf[ID].Msg==SoA || FrameOf[ID].Msg==ASnd || FrameOf[ID].Msg==AMNI
Dtgrm? FrameLossDetectIfSoA(), TcycClockResetIfAMNI()
FrameOf[ID].Msg==SoA || FrameOf[ID].Msg==AMNI
Dtgrm? FrameLossDetectIfSoA(), TcycClockResetIfAMNI()
FrameOf[ID].Msg==SoC || FrameOf[ID].Msg==PRes
Dtgrm? FrameLossDetectAndTcycClockResetIfSoC()


randID: idNode
DelayClock==PResTimeout
[PollProg[PollProgStep]] &&
nextIsSoA()
Dtgrm!
initPollingParam(),
ASndID:=randID,
setParamOfFrame(SoA)
ID==FrameOf[ID].Addr &&
DelayClock==tSoA2ASnd[ID] &&
TcycClock <=Tcyc
Dtgrm!
setParamOfFrame(ASnd)
AMN_Delaying_PRes
DelayClock <=
tPRes2PRes[ID]
DelayClock==tPRes2PRes[ID]
Dtgrm!
setParamOfFrame(PRes),
DelayClock:=0
AMN_Delaying_SoA
DelayClock <=
tPRes2SoA[ID]
randID: idNode
DelayClock==tPRes2SoA[ID]
Dtgrm!
ASndID:=randID,
setParamOfFrame(SoA),
DelayClock=0
AMN_Waiting_Or_Delaying_ASnd
DelayClock <=
((ID!=FrameOf[ID].Addr)?
ASndTimeout[ID]:tSoA2ASnd[ID])
TcycClock >Tcyc
AMN_TcycError
AllStatesAMNFailureEnabled &&
AMNFailures<AMNFailuresMax
updateAMNFailureParam()
AMNFailures<
AMNFailuresMax &&
DelayClock==tPRes2SoA[ID]
updateAMNFailureParam()
AllStatesAMNFailureEnabled &&
AMNFailures<AMNFailuresMax
updateAMNFailureParam()
RMN_Failing
Dtgrm?
SMNFailures<
SMNFailuresMax &&
DelayClock==tPReq2PRes[ID]
updateSMNFailureParam()
SMNFailures<
SMNFailuresMax &&
TcycClock==tSwitchOver
updateSMNFailureParam()
AllStatesSMNFailureEnabled &&
SMNFailures<SMNFailuresMax
updateSMNFailureParam()
SMN_Delaying_PRes
DelayClock <=
tPReq2PRes[ID]
&&
TcycClock <=
tSwitchOver
DelayClock==tPReq2PRes[ID]
Dtgrm!
setParamOfFrame(PRes)
SMN_Waiting_SoA
TcycClock <=
tSwitchOver
FrameOf[ID].Msg==SoA &&
FrameOf[ID].Addr==ID
Dtgrm?
DelayClock:=0
(FrameOf[ID].Msg==SoA &&
FrameOf[ID].Addr!=ID) ||
FrameOf[ID].Msg==AMNI
Dtgrm?
TcycClockResetIfAMNI()
FrameOf[ID].Msg==SoC
Dtgrm?
TcycClock:=0,
FramelossDetected++
FrameOf[ID].Msg==PRes
Dtgrm?
SMN_Delaying_ASnd
DelayClock <=
tSoA2ASnd[ID]
&&
TcycClock <=
tSwitchOver
DelayClock==tSoA2ASnd[ID]
Dtgrm!
setParamOfFrame(ASnd)



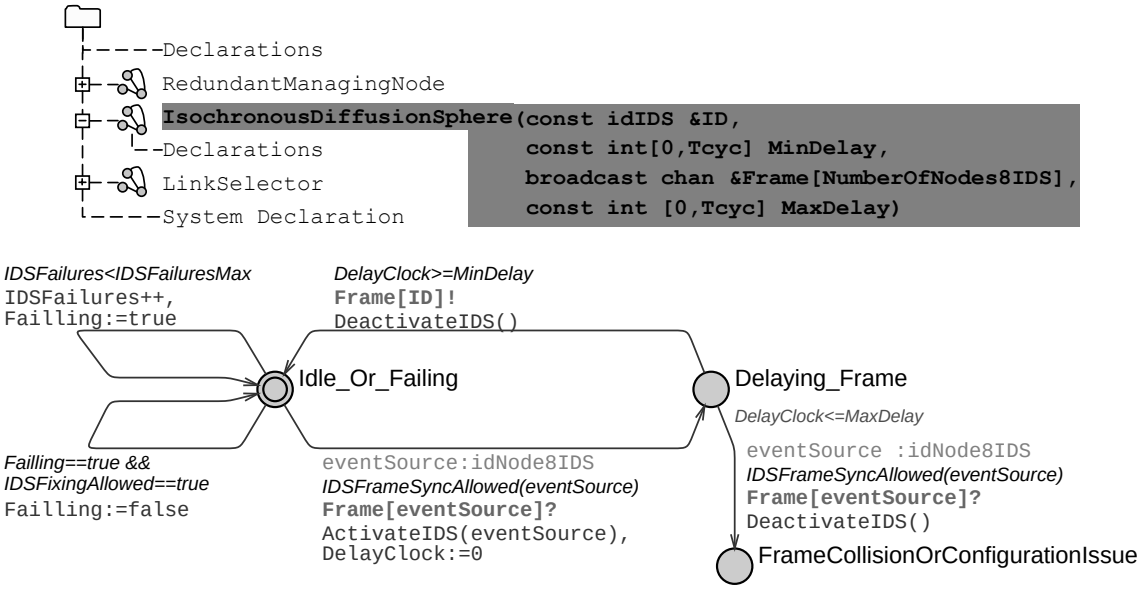

```
////////////////////////////////////////////////////////////////////////////////
//                          MODEL OF A NODE DECLARATIONS CODE                 //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES CODE_____________________________________*/
clock TcycClock;
clock DelayClock;
bool isAMN=false;
meta int FramelossDetected=0;

/*_______________________________FUNCTIONS CODE_____________________________________*/

/*----------------Functions used in GUARDS CODE-----------------------------------*/
bool expectedPRes()
{ if (FrameOf[ID].Msg==PRes  &&
FrameOf[ID].Addr==PollProg[PollProgStep])
{return true;}
else {return false;}
}
bool nextIsPReq()
{ if(((PollProgStep+1)<PollProgSize) && ((PollProg[PollProgStep+1]!= ID) ||
(PollProg[PollProgStep+1]== ID && (PollProgStep+1)&lt;PollProgSize-1)))
{return true;}
else {return false;}
}
bool nextIsAMNPRes()
{ if( (PollProgStep+1==PollProgSize) &&
(not (forall (SlotNumber : int [0,PollProgSize-1])(PollProg[SlotNumber]!=ID)))
||
((PollProgStep+2==PollProgSize) && (PollProg[PollProgStep+1]==ID)) )
{return true;}
else {return false;}
}
bool nextIsSoA()
{ if( (PollProgStep+1==PollProgSize) &&
(forall (SlotNumber : int [0,PollProgSize-1])(PollProg[SlotNumber]!=ID)))
{return true;}
else {return false;}
}
bool OwnPResPlanned()
{ return (not (forall (SlotNumber : int [0,PollProgSize-1])
(PollProg[SlotNumber]!=ID)));}
```

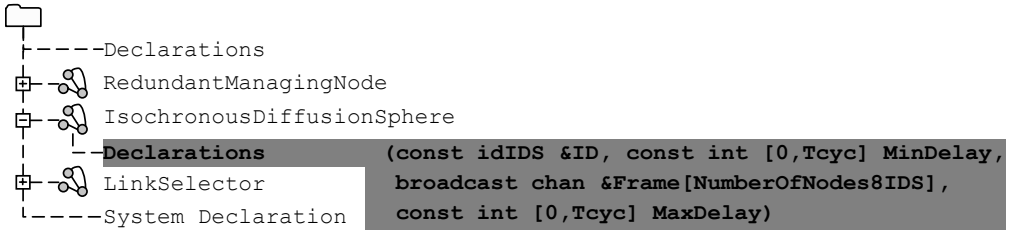
```
////////////////////////////////////////////////////////////////////////////////
//                       MODEL OF A NODE DECLARATIONS COMMENTS                //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES COMMENTS_____________________________________*/
// Node own clock to monitor EPL cycle time
// Node own clock to model node's delays and monitor timeouts
// updated by the model to give overall state: [AMN] or [SMN, inactive, failling]
// updated by the model when detecting a "loss of frame" respect to EPL cycle

/*_______________________________FUNCTIONS COMMENTS_____________________________________*/

/*----------------Functions used in GUARDS COMMENTS--------------------------------*/
// Checks if the frame received after having sent a PReq is a PRes from polled SMN




// Checks if more nodes need to be polled, given the polling program




// Checks if next frame is the PRes from AMN, given the polling program







// Checks if next frame is the SoA from AMN, given the polling program




// Checks a PRes from SMN, is planned in the polling program
```

```
/*----------------Functions used in UPDATES CODE---------------------------------*/
void initFrameSyncMatrix()
{ if ( forall (line: int [0,NumberOfNodes8IDS-1])
(forall (column: int [0,NumberOfIDS-1])
(ListensToIDS[line][column]==false ) ) )
{ for (column : int [0,NumberOfIDS-1])
{ for (line : idNode)
{ListensToIDS[line][column]=NodePortsAssignement[line][column];}
for (line : int[0,NumberOfIDS-1])
{ListensToIDS[NumberOfNodes+line][column]=LinksBetweenIDS [line][column];}
}
}
}
void initPollingParam()
{ PollProgStep=0; DelayClock=0; }

void updatePollingParam(bool caze)
{ if (caze==0) { PollProgStep=
(PollProg[PollProgStep]== ID)?(PollProgStep+1):PollProgStep;}
else
{ if (PollProg[PollProgStep+1]==ID){PollProgStep=PollProgStep+2;}
else {PollProgStep++;}
DelayClock=0;
}
}
void TcycClockResetIfAMNI()
{ if (FrameOf[ID].Msg==AMNI) {TcycClock=0;} }

void FrameLossDetectIfSoA()
{ if (FrameOf[ID].Msg==SoA) {FramelossDetected++;} }

void FrameLossDetectAndTcycClockResetIfSoC()
{ if (FrameOf[ID].Msg==SoC) {TcycClock:=0;FramelossDetected++;}}

void updateAMNFailureParam()
{ isAMN=false; PollProgStep=0;AMNFailures++;}

void updateSMNFailureParam()
{ isAMN=false; SMNFailures++;}

void setParamOfFrame (idMsg MsgType)
{ FrameOf[ID].Msg:=MsgType;
if (MsgType==SoC){ FrameOf[ID].Addr:=ID;
 if (PollProgStep<NumberOfNodes && PollProg[PollProgStep]== ID)
{ PollProgStep++;}
initPollingParam();}
else if (MsgType==PReq) {FrameOf[ID].Addr:=PollProg[PollProgStep]; DelayClock=0; }
else if (MsgType==PRes||MsgType==ASnd) { FrameOf[ID].Addr:=ID; }
else if (MsgType==SoA ) { if ( AsndLimiterEnabled &&
(forall(i : int [0,NumberOfAsndEnabled-1]) ASndSenderIDLimiter[i]!=ASndID))
{ FrameOf[ID].Addr:=ASndSenderIDLimiter[0];}
else { FrameOf[ID].Addr:=ASndID;}
ASndID=0;}
else if (MsgType==AMNI) { FrameOf[ID].Addr:=ID; isAMN:=true; }
else { FrameOf[ID].Addr:=1000; }
}
////////////////////////////////////////////////////////////////////////////////
//                     END OF MODEL OF A NODE DECLARATIONS CODE                 //
////////////////////////////////////////////////////////////////////////////////
```

```
/*----------------Functions used in UPDATES-COMMENTS-------------------------------*/
// Initialize matrix allowing "frame-event" synchronization
//  If matrix ListensToIDS is not initialzed by the model (all values to false)



//  Itinialize LS<->IDS allowed synchronization

//  Itinialize IDS<->IDS allowed synchronization



// Initialize to the begining of the polling program


// Updates the step achieved in the polling program
// "case" is a reserved keyword







// Conditionnal TcyClock reset


// Conditionnal FramelossDetected counter update


// Conditionnal FramelossDetected counter update and TcyClock reset


// Updates model parameters when going to failure from AMN overall state


// Updates model parameters when going to failure from SMN overall state


// Update variables frame id associated with emitted frame event
```

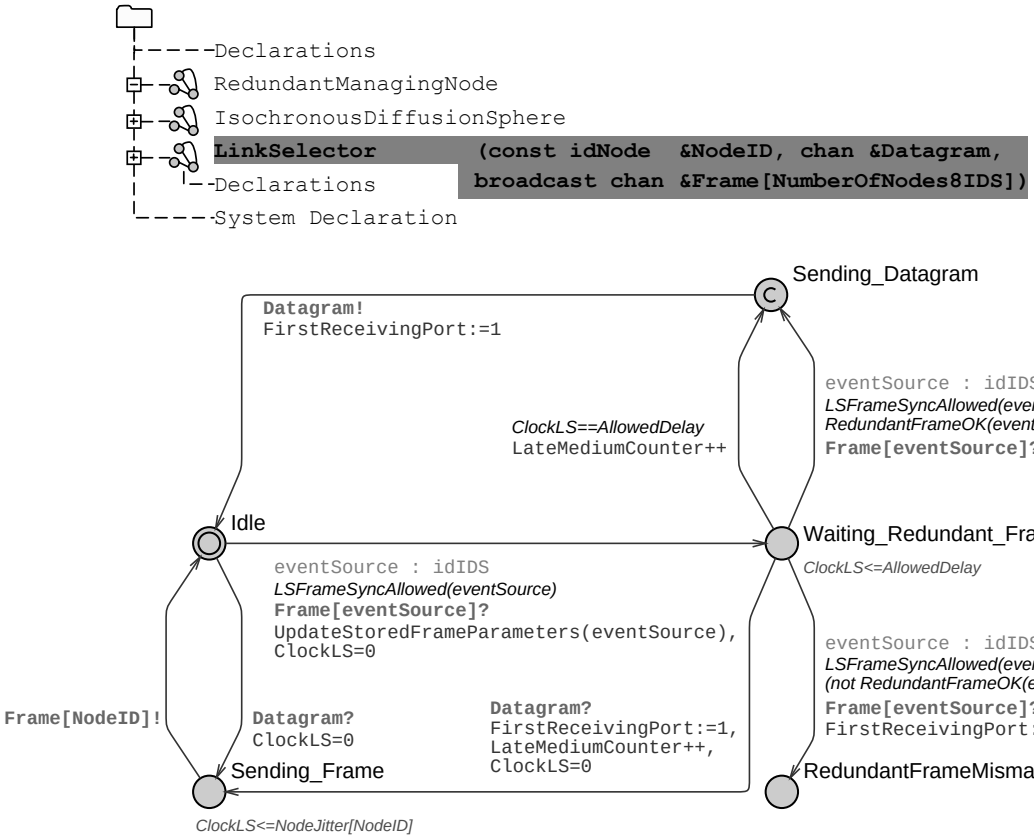
```
////////////////////////////////////////////////////////////////////////////////
//                       END OF MODEL OF A NODE DECLARATIONS                    //
////////////////////////////////////////////////////////////////////////////////
```

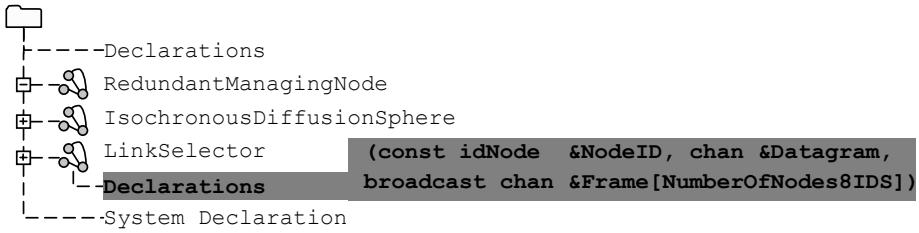

Declarations
RedundantManagingNode
IsochronousDiffusionSphere(const idIDS &ID,
const int[0,Tcyc] MinDelay,
broadcast chan &Frame[NumberOfNodes8IDS],
const int [0,Tcyc] MaxDelay)
Declarations
LinkSelector
System Declaration
IDSFailures<IDSFailuresMax
IDSFailures++,
Failling:=true
DelayClock>=MinDelay
Frame[ID]!
DeactivateIDS()
Idle_Or_Failing
Delaying_Frame
DelayClock<=MaxDelay
eventSource :idNode8IDS
IDSFrameSyncAllowed(eventSource)
Frame[eventSource]?
DeactivateIDS()
FrameCollisionOrConfigurationIssue
Failling==true &&
IDSFixingAllowed==true
Failling:=false
eventSource:idNode8IDS
IDSFrameSyncAllowed(eventSource)
Frame[eventSource]?
ActivateIDS(eventSource),
DelayClock:=0




```
////////////////////////////////////////////////////////////////////////////////
//                    MODEL OF AN IDS DECLARATIONS CODE                       //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES CODE_____________________________________*/
clock DelayClock;
bool SavListenedVector[NumberOfNodes8IDS];
bool Failling=false;
/*_______________________________FUNCTIONS CODE_____________________________________*/
/*----------------Functions used in GUARDS CODE-------------------------------------*/
bool IDSFrameSyncAllowed(idNode8IDS synctransmitterID)
{ if ( (not Failling) &&
(((synctransmitterID<NumberOfNodes) &&
(NodePortsAssignement[synctransmitterID][ID-NumberOfNodes]!=0))
|| ((synctransmitterID>=NumberOfNodes) &&
(ListensToIDS[ID][synctransmitterID-NumberOfNodes])))
{ return true;}
else { return false;}
}

/*----------------Functions used in UPDATES CODE------------------------------------*/
void ActivateIDS(idNode8IDS SyncTransmitterID)
{
for (j : idNode8IDS)
{ SavListenedVector[j]=ListensToIDS[j][ID-NumberOfNodes];
if (ListensToIDS[j][ID-NumberOfNodes]==true)
{ if ((SyncTransmitterID>=NumberOfNodes) &&
(ListensToIDS[j][SyncTransmitterID-NumberOfNodes]==true))
{ ListensToIDS[j][ID-NumberOfNodes]=false; }
if((SyncTransmitterID<NumberOfNodes) && (j>=NumberOfNodes))
{ if (NodePortsAssignement[SyncTransmitterID][j-NumberOfNodes]!=0)
{ListensToIDS[j][ID-NumberOfNodes]=false; }
}
}
}
ListensToIDS[SyncTransmitterID][ID-NumberOfNodes]:=false;

FrameOf[ID]=FrameOf [SyncTransmitterID];
}

void DeactivateIDS()
{ for (j : idNode8IDS)
{ ListensToIDS[j][ID-NumberOfNodes]=SavListenedVector[j]; SavListenedVector[j]=0; }
}
////////////////////////////////////////////////////////////////////////////////
//                    END OF MODEL OF AN IDS DECLARATIONS CODE                //
////////////////////////////////////////////////////////////////////////////////
```

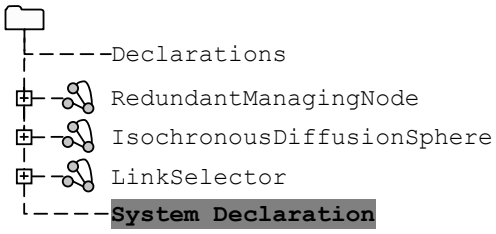
```
////////////////////////////////////////////////////////////////////////////////
//                   MODEL OF AN IDS DECLARATIONS COMMENTS                    //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES COMMENTS_____________________________________*/
// IDS own clock to indroduce delays in event transmission
// vector store who was allowed to synchonize with IDS
// Store IDS failure state
/*_______________________________FUNCTIONS COMMENTS_____________________________________*/
/*----------------Functions used in GUARDS COMMENTS--------------------------------*/
// Conditions to allow synchonisations with frame events from IDS or LS

// Allowed to synchonizes with frame events from LS

// OR Allowed to synchonizes with frame events from IDS




/*----------------Functions used in UPDATES COMMENTS-------------------------------*/
// Modify who is allowed to synchonize with IDS to prevent "echo(?)" back
//


// Considered ID synchronizes with me

// If sender is an IDS (not a LS) and considered ID synchronizes with it
// considered ID is not synchronizing with me for a moment
// If sender is a LS and considered ID is owned by an IDS
// If considered ID also synchronizes with sender
// considered ID is not synchronizing with me for a moment



// synchronization is prevented to go back to sender
// sender won't ear my next synchronization
// Copy frame parameters from IDS or LS emitting the event


// Modify who is allowed to synchonize with IDS to previous values



////////////////////////////////////////////////////////////////////////////////
//                   END OF MODEL OF AN IDS DECLARATIONS COMMENTS              //
////////////////////////////////////////////////////////////////////////////////
```


Declarations
RedundantManagingNode
IsochronousDiffusionSphere
LinkSelector
(const idNode &NodeID, chan &Datagram,
broadcast chan &Frame[NumberOfNodes8IDS])
Declarations
System Declaration
Sending_Datagram
Datagram!
FirstReceivingPort:=1
eventSource : idIDS
LSFrameSyncAllowed(eventSource) &&
RedundantFrameOK(eventSource)
Frame[eventSource]?
ClockLS==AllowedDelay
LateMediumCounter++
Idle
Waiting_Redundant_Frame
ClockLS<=AllowedDelay
eventSource : idIDS
LSFrameSyncAllowed(eventSource)
Frame[eventSource]?
UpdateStoredFrameParameters(eventSource),
ClockLS=0
eventSource : idIDS
LSFrameSyncAllowed(eventSource) &&
(not RedundantFrameOK(eventSource))
Frame[eventSource]?
FirstReceivingPort:=1
Frame[NodeID]!
Datagram?
ClockLS=0
Datagram?
FirstReceivingPort:=1,
LateMediumCounter++,
ClockLS=0
Sending_Frame
RedundantFrameMismatchIssue
ClockLS<=NodeJitter[NodeID]




```
////////////////////////////////////////////////////////////////////////////////////
//                          THE MODEL OF A LS DECLARATIONS CODE                   //
////////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES CODE_____________________________________*/
clock ClockLS;
const int AllowedDelay=50;
meta int LateMediumCounter=0;
int [1,2] FirstReceivingPort;
/*_______________________________FUNCTIONS CODE_____________________________________*/
/*----------------Functions used in GUARDS CODE---------------------------------*/
bool LSFrameSyncAllowed(idIDS synctransmitterID)
{
if  ((ListensToIDS[NodeID][synctransmitterID-NumberOfNodes]==true) &&
((FrameOf[synctransmitterID].Msg==SoC) ||
  (FrameOf[synctransmitterID].Addr==NodeID) ||
  (FrameOf[synctransmitterID].Msg==PRes) ||
  (FrameOf[synctransmitterID].Msg==SoA) ||
  (FrameOf[synctransmitterID].Msg==ASnd) ||
  (FrameOf[synctransmitterID].Msg==AMNI)))
{ return true;}
else{return false;}
}
bool RedundantFrameOK(idIDS synctransmitterID)
{
if ( NodePortsAssignement[NodeID][synctransmitterID-NumberOfNodes]!=
(FirstReceivingPort) && FrameOf[NodeID]==FrameOf[synctransmitterID])
{return true; }
else {return false; }
}
/*----------------Functions used in UPDATES CODE--------------------------------*/
void UpdateStoredFrameParameters(idIDS SyncTransmitterID)
{
FrameOf[NodeID]:=FrameOf [SyncTransmitterID];
FirstReceivingPort:=NodePortsAssignement[NodeID][SyncTransmitterID-NumberOfNodes];
}
////////////////////////////////////////////////////////////////////////////////////
//                        END OF MODEL OF A LS DECLARATIONS CODE                  //
////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////
//                   THE MODEL OF A LS DECLARATIONS COMMENTS                  //
////////////////////////////////////////////////////////////////////////////////
/*_______________________________VARIABLES COMMENTS_________________________________*/
// LS own clock to monitor delays between media
// Allowed delay between the two ports before considering a late medium error
// updated by the model when FrameSelectiondelay is achieved.
// updated by the model to check recpetion on both ports
/*_______________________________FUNCTIONS COMMENTS_________________________________*/
/*----------------Functions used in GUARDS-----------------------------------------*/
// Conditions to allow synchonisations with frame events from IDS

// LS is allowed to sync with this IDS by topoly configuration and ste of IDS
// Frame type is broadcasted or node is addressed







// Check if the received event is the redundant of the previous one


// Event is received on the redundant port and frame parameters are the same



/*----------------Functions used in UPDATES COMMENTS-------------------------------*/
// Updates LS+node parameters on frame event synchronisation

// Copy frame parameters from IDS emitting the event
// store active port

////////////////////////////////////////////////////////////////////////////////
//                   END OF MODEL OF A LS DECLARATIONS COMMENTS                //
////////////////////////////////////////////////////////////////////////////////
```




```
////////////////////////////////////////////////////////////////////////////////
//                          TEMPLATE INSTANTIATIONS CODE                      //
////////////////////////////////////////////////////////////////////////////////
//
RMN (const idNode ID) =
RedundantManagingNode ( ID,
DatagramSync[ID],
Tcyc+(ID+1)*SliceTime,
Tcyc-(ID+1)*SliceTime
);
//
LS  (const idNode ID) =
LinkSelector ( ID,
FrameSync,
DatagramSync[ID]
);
//
IDS (const idIDS ID) =
IsochronousDiffusionSphere ( ID,
FrameSync,
IDSLatency[ID-NumberOfNodes]+IDSJitter[ID-NumberOfNodes]/2,
IDSLatency[ID-NumberOfNodes]-IDSJitter[ID-NumberOfNodes]/2
);

////////////////////////////////////////////////////////////////////////////////
//                             SYSTEM DECLARATION CODE                        //
////////////////////////////////////////////////////////////////////////////////
system RMN,
LS,
IDS
;
```

```
////////////////////////////////////////////////////////////////////////////////
//                          TEMPLATE INSTANTIATIONS COMMENTS                  //
////////////////////////////////////////////////////////////////////////////////
//
// RMN instantiation


// Set time before switch over from SMN to AMN
// Set time to used to abide by SoC regularity when switching to AMN

//
// LS instantiation




//
// IDS instantiation


// Set IDS maximum forwarding delay
// Set IDS minimum forwarding delay


////////////////////////////////////////////////////////////////////////////////
//                                  SYSTEM DECLARATION COMMENTS               //
////////////////////////////////////////////////////////////////////////////////
```

Copie d'écran de l'application EPLDesigner

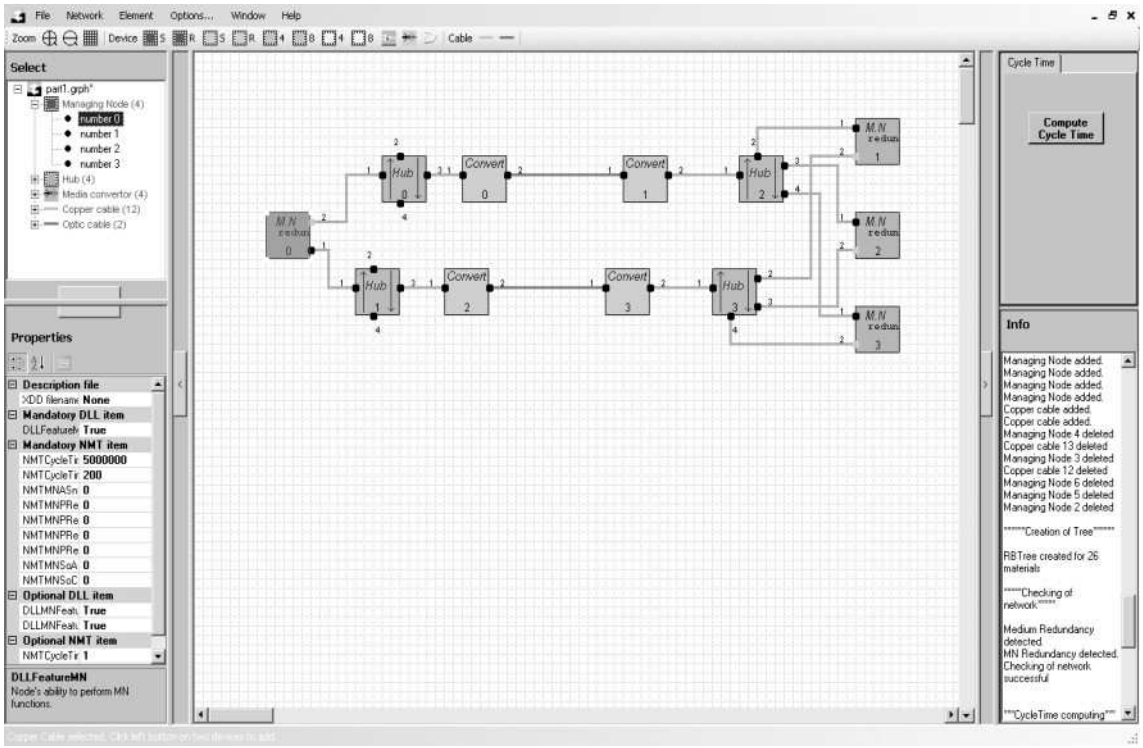


FIGURE 9 – EPLDesigner.

Références Scientifiques

[1] Gaëlle Marsal. *Evaluation of time performances of Ethernet-Based Automation Systems by simulation of High-level Petri Nets*. Thèse de doctorat, Laboratoire Universitaire de Recherche en Production Automatisée, ENS Cachan (France) and Univ. of Kaiserslautern (Germany), 2006.

[2] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*, pages 87–101. Springer-Verlag London, UK, 1991.

[3] F. Scheler and W. Schröder-Preikschat. Time-triggered vs. event-triggered : A matter of configuration ? In *Proceedings of the GI/ITG Workshop on Non-Functional Properties of Embedded Systems*, pages 107–112. VDE Verlag, 2006.

[4] Stéphane Pailler. *Analyse Hors Ligne d'Ordonnançabilité dÁpplications Temps Réel comportant des Tâches Conditionnelles et Sporadiques*. Thèse de doctorat, Université de Poitiers - École Nationale Supérieure de Mécanique et d´Aérotechnique, 2006.

[5] Francisco Vasques de Carvalho. *Sur l'intégration de mécanismes dórdonnancement et de communication dans la sous-couche MAC de réseaux locaux temps-réel*. Thèse de doctorat, Université Paul Sabatier de Toulouse, 1996.

[6] E.R. Dougherty and P.A. Laplante. *Introduction to real-time imaging*. IEEE Press New York, 1995.

[7] M. Felser. Real-time ethernet - industry prospective. *Proceedings of the IEEE*, 93(6) :1118–1129, June 2005.

[8] J. Arlat, Y. Crouzet, Y. Deswarte, J.C. Fabre, J.C. Laprie, and D. Powell. *Fault Tolerance*, chapter Encyclopedia of Computer Science and Information Systems, Part 1 : The Technological Dimension of Information Systems, Section 2 : Architecture and Systems, pages 240 – 270. Vuibert, 2006. ISBN : 2-7117-4846-4. (in French).

[9] J. Jasperneite and P. Neumann. How to guarantee real-time behavior using ethernet. In *Proc. of 11th IFAC Symp. on Information Control Problems in Manufacturing*, page 115–140, 2004.

[10] Jason J. Scarlett and Robert W. Brennan. Re-evaluating event-triggered and time-triggered systems. In *ETFA '06* [69], pages 655–661.

[11] J.-D. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6) :1102–1117, June 2005.

[12] J.-P. Thomesse. Fieldbus technology in industrial automation. *Proceedings of the IEEE*, 93(6) :1073–1101, 2005.

[13] Y. Song, A. Koubaa, and F. Simonot. Switched ethernet for real-time industrial communication : modelling and message buffering delay evaluation. In *WFCS '02* [70], pages 27–35.

[14] Jean-Philippe Georges, Thierry Divoux, and Eric Rondeau. Validation of the network calculus approach for the performance evaluation of switched ethernet based industrial communications. In *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic*, 2005.

[15] B.Y. Choi, S. Song, N. Birch, and J. Huang. Probabilistic approach to switched ethernet for real-time control applications. In USA IEEE Computer Society Washington, DC, editor, *Proceedings of the Seventh International Conference on Real-Time Systems and Applications (RTCSA'00)*, page 384, 2000.

[16] J. Jasperneite, P. Neumann, M. Theis, and K. Watson. Deterministic real-time communication with switched ethernet. In *WFCS '02* [70], pages 11–18.

[17] Naveen Kalappa, Kristen Acton, Marco Antolovic, Siddharth Mantri, Jonathan Parrott, Jonathan E. Luntz, James R. Moyne, and Dawn M. Tilbury. Experimental determination of real time peer to peer communication characteristics of ethernet/ip. In *ETFA '06* [69], pages 1061–1064.

[18] Jean-Philippe Georges, Nicolas Krommenacker, Thierry Divoux, and Eric Rondeau. A design process of switched ethernet architectures according to real-time application constraints. *Engineering Applications of Artificial Intelligence*, 19(3) :335–344, 2006.

[19] B. Denis, S. Ruel, J.M. Faure, G. Frey, and G. Marsal. Measuring the impact of vertical integration on response times in ethernet fieldbuses. In *ETFA '07* [71], pages 532–539.

[20] Juergen Jasperneite, Markus Schumacher, and Karl Weber. Limits of increasing the performance of industrial ethernet protocols. In *ETFA '07* [71], pages 17–24.

[21] Paulo Pedreiras, Luís Almeida, and Paolo Gai. The ftt-ethernet protocol : Merging flexibility, timeliness and efficiency. In *Proceedings of 14th Euromicro Conference on Real-Time Systems*, page 152. IEEE Computer Society, June 2002.

[22] TC Chiueh. Rether : A software-only real-time ethernet for plc networks. *usenix Association Proceedings of the Workshop on Embedded Systems*, page 45–53, March 1999.

[23] J.M. Martínez, M.G. Harbour, and J.J. Gutiérrez. Rt-ep : Real-time ethernet protocol for analyzable distributed applications on a minimum real-time posix kernel. In *Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age, RTLIA*, 2003.

[24] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink. Rtnet-a flexible hard real-time networking framework. In *Proc. of 10th IEEE Conf. on Emerging Technologies and Factory Automation*, 2005.

[25] L. Seno and S. Vitturi. A simulation study of ethernet powerlink networks. In *ETFA '07* [71], pages 740–743.

[26] P. Ferrari, A. Flammini, and S. Vitturi. Performance analysis of profinet networks. *Computer Standards & Interfaces*, 28(4) :369–385, 2006.

[27] J. Jasperneite and E. Elsayed. Investigations on a distributed time-triggered ethernet realtime protocol used by profinet. In *3rd International Workshop on Real-Time Networks (RTN 2004) Catania, Italy*, June 30-July 2 2004.

[28] D. Heffernan and P. Doyle. Research article time-triggered ethernet based on ieee 1588 clock synchronisation. *Assembly Automation*, 24(3) :264–269, 2004.

[29] M. Felser and T. Sauter. Standardization of industrial ethernet - the next battlefield ? In *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, pages 413–420, 22-24 Sept. 2004.

[30] G. Prytz. Redundancy in industrial ethernet networks. In *WFCS '06* [72], pages 380–385.

[31] Z. Wang, Y. Song, J. Chen, and Y. Sun. Real-time characteristics of ethernet and its improvement. In *Proc. of the 4th World Congress on Intelligent Control and Automation*, page 1311–1318, 2002.

[32] H. Kirrmann and D. Dzung. Selecting a standard redundancy method for highly available industrial networks. In *WFCS '06* [72], pages 386–390.

[33] L. Ruiz. *Réseaux de terrain à hautes performances basés sur le modèle Producteur-Distributeur-Consommateur*. Thèse de doctorat, Département d'Informatique. Lausanne : Ecole Polytechnique Fédérale de Lausanne, 1996.

[34] T. Skeie, S. Johannessen, C. Brunner, and A.B.B.C. Res. Ethernet in substation automation. *Control Systems Magazine, IEEE*, 22(3) :43–51, 2002.

[35] D. Witsch, B. Vogel-Heuser, J.M. Faure, and G. Marsal. Performance analysis of industrial ethernet networks by means of timed model-checking. In *12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006*, pages 101–106, Saint-Etienne (France), May 2006.

[36] J.-M. Machado, B. Denis, and J.-J. Lesage. Formal verification of industrial controllers : With or without a plant model ? In *CONTROLO'2006 – The 7th Portuguese Conference on Automatic Control, Lisboa, Portugal, September, 11-13th, 2006, (6 p.)*, 2006.

[37] I. Moon. Modeling programmable logic controllers for logic verification. *Control Systems Magazine,IEEE*, 14(2) :53–59, 1994.

[38] G. Frey and L. Litz. Formal methods in plc programming. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, 2000.

[39] J.M. Roussel and B. Denis. Safety properties verification of ladder diagram programs. *Journal européen des systèmes automatisés*, 36(7) :905–917, 2002.

[40] Béatrice Bérard, Michel Bidoit, Alain Finkel, Fran¸cois Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[41] S. Campos, E.M. Clarke, and M. Minea. Symbolic techniques for formally verifying industrial systems. *Science of computer programming*, 29(1-2) :79–98, 1997.

[42] Emmanuel Fleury. *Les automates temporisés avec mises à jour*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2002.

[43] Sylvain Peyronnet. *Model checking et vérification probabiliste*. Thèse de doctorat, Université Paris-Sud, Décembre 2003.

[44] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

[45] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited : on branching versus linear time temporal logic. *Journal of the ACM*, 33(1) :151–178, 1986.

[46] M. Fruth. Probabilistic model checking of contention resolution in the ieee 802.15.4 low-rate wireless personal area network protocol. In *Proc. 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'06)*, 2006.

[47] Vincent Gourcuff. *Représentations formelles efficaces pour l' aide à la certification de contrôleurs logiques industriels.* Thèse de doctorat, Laboratoire Universitaire de Recherche en Production Automatisée, ENS Cachan, France, dec 2007.

[48] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL-a tool suite for automatic verification of real-time systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems*, page 232–243, 1995.

[49] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1995. isbn = 3-540-61155-X.

[50] Nicolas Markey and Philippe Schnoebelen. Symbolic model checking for simply-timed systems. In Yassine Lakhnech and Sergio Yovine, editors, *Proceedings of the Joint Conferences Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 102–117, Grenoble, France, sep 2004. Springer.

[51] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex : A tool for reachability analysis of complex systems. In *Proc. 13th Int. Conf. Computer Aided Verification (CAV'2001), Paris, France*, volume 2102, pages 368–372, 2001.

[52] T.A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. Hytech : the next generation. *rtss*, 00 :56, 1995.

[53] G. Frehse. PHAVer : Algorithmic verification of hybrid systems past HyTech. *HSCC*, pages 258–273, 2005.

[54] H. Song, K.J. Compton, and W.C. Rounds. SPHIN : A model checker for reconfigurable hybrid systems based on SPIN. *Electronic Notes in Theoretical Computer Science*, 145 :167–183, 2006.

[55] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 2.0 : A tool for probabilistic model checking. In *QEST*, pages 322–323. IEEE Computer Society, 2004.

[56] T. Herault, L.R.I.U. Paris-Sud, R. Lassaigne, and S. Peyronnet. APMC 3.0 : Approximate Verification of Discrete and Continuous Time Markov Chains. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of SysTems (QEST) 2006, California, USA*, Sept 2006.

[57] Charfi Faiez. Une approche dínterfaçage de cod à uppaal pour la spécification et la vérification des systèmes temps réel. Master's thesis, Diplôme d'Etudes Approfondies en Informatique – Faculté des sciences de Tunis, Septembre 2003.

[58] Béatrice Bérard, Patricia Bouyer, and Antoine Petit. Analysing the PGM protocol with Uppaal. *International Journal of Production Research*, 42(14) :2773–2791, July 2004.

[59] Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 165–172. IEEE Computer Society, 2000.

[60] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, April 1994.

[61] Patricia Bouyer. *Modèles et algorithmes pour la vérification des systèmes temporisés.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, apr 2002.

[62] G. Behrmann, A. David, and K.G. Larsen. A tutorial on Uppaal. *Revised Lectures of the International School on Formal Methods for the Design of Real-Time Systems,(SFM-RT'04)*, 3185 :200–236, 2004.

[63] Houda Bel mokadem. *Vérification des propriétés temporisées des automates programmables industriels.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, sep 2006.

[64] Philippe Schnoebelen, Béatrice Bérard, Michel Bidoit, François Laroussinie, and Antoine Petit. *Vérification de logiciels : techniques et outils du model-checking.* Vuibert, April 1999.

[65] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, 1993.

[66] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logics of Programs*, pages 52–71, 1981.

[67] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 169–180. ACM Press New York, NY, USA, 1982.

[68] Zulema Juarez Orozco. *Vérification de propriétés quantitatives des systèmes logiques par model-checking hybride.* Thèse de doctorat, Laboratoire Universitaire de Recherche en Production Automatisée, ENS Cachan, France, jun 2008.

[69] *Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2006, September 20-22, 2006, Diplomat Hotel Prague, Czech Republic.* IEEE, 2006.

[70] *Proceedings of 4th IEEE International Workshop on Factory Communication Systems.* IEEE, 2002.

[71] *Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2007.* IEEE, 2007.

[72] *Proceedings of 6th IEEE International Workshop on Factory Communication Systems.* IEEE, June 27, 2006.

Références Techniques

[73] IEEE. Standard 610.12-1990 : Glossary of software engineering terminology, 1999.

[74] SC 65C. IEC61784-3 : Industrial communication networks - profiles - part 3 : Functional safety fieldbuses - general rules and profile definitions, 2007.

[75] SC 65A. IEC61508-4 : Functional safety of electrical / electronic / programmable electronic safety-related systems – part 4 : Definitions and abbreviations, 1998.

[76] EPSG. Ethernet powerlink v2.0 communication profile specification. draft standard,version 1.0.0, 2006.

[77] Hirschmann Automation and Control. http ://www.hirschmann-ac.de/.

[78] EPSG. http ://www.ethernet-powerlink.org/.

[79] Profinet International. http ://www.profibus.com/pn/.

[80] Ethernet for Control Automation Technology (EtherCAT) Technology Group. http ://www.ethercat.org/.

[81] SC 65C. IEC62439 : High availability automation networks, 2007.

[82] IEEE. Standard 1588-2008 : Precision clock synchronization protocol for networked measurement and control systems, 2008.

[83] SC 65C. IEC61588 : Precision clock synchronization protocol for networked measurement and control systems, 2004.

[84] SC 65C. IEC61784-2 : Industrial communication networks - profiles - part 2 : Additional fieldbus profiles for real-time networks based on iso/iec 8802-3, 2007.

[85] SC 65C. IEC61158 : Industrial communication networks - fieldbus specifications, 2007.

[86] EPSG. Ethernet powerlink v2.0 high availability specification. working standard proposal,version 0.1.3, 2007.

[87] TC 56-Dependability. IEC61078 : Analysis techniques for dependability –reliability block diagram and boolean methods, 2006.

[88] G. Huet, G. Kahn, and C. Paulin-Mohring. The coq proof assistant : A tutorial (version 7.4). Technical report, Inria, 2003.

[89] IEEE. Standard 802.15.4-2006 : Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans), 2006.

[90] Uppaal. http ://www.uppaal.com/.

Résumé : Les travaux présentés dans ce mémoire s'intéressent aux mécanismes de redondance spécifiés par un protocole de réseau de terrain sur Ethernet. L'objectif est de valider la spécification par rapport à des exigences de disponibilité. Le contexte industriel des travaux nous a amenés à :

- privilégier une validation par vérification formelle. Le model-checking temporel a été retenu. En effet, le caractère critique des applications industrielles pour lesquelles doit être intégré le protocole impose une vérification exhaustive des propriétés. Les nombreux paramètres temporels permettant de décrire le fonctionnement du protocole nous ont amenés à favoriser une technique prenant en compte le temps.
- proposer une modélisation par automates temporisés modulaire ainsi qu'une méthode d'instanciation. Celles-ci permettent d'adapter facilement le modèle à vérifier à une architecture de réseau de terrain envisagée lors de la phase d'étude d'une affaire.
- proposer des abstractions qui favorisent la vérification du modèle par le moteur de model-checking temporel. Les propriétés vérifiées traduisent l'aptitude des mécanismes de redondance à compenser une défaillance du médium ou de l'animation des échanges.

Afin d'illustrer la pertinence de ces propositions, la méthode d'instanciation est appliquée à deux architectures et une campagne de vérifications est menée et analysée.

Mots-clés : vérification formelle par model-checking temporel, Ethernet PowerLink (EPL), modélisation, abstraction, Uppaal, disponibilité

Abstract : This work deals with an Ethernet based protocol that specifies redundancy mechanisms. The objective is to check the specification respect to availability properties. The industrial context of this work led us to :

- give priority to a formal method. Temporal model-checking has been selected. Indeed the protocol must be used in critical industrial control systems. Therefore a thorough verification is necessary. The protocol description depends on many temporal parameters. Consequently, a technique taking time into account has been preferred.
- put forward a method to instantiate our model. This timed automata based model has been designed to be modular. Thus the modelling of any network architecture is possible without requiring any modelling skills from the engineer.
- put forward abstractions in order to improve the model-checking time and memory consumption. Checked properties describe the redundancy mechanisms capability to keep communications working event in case of medium or end device failure.

In order to illustrate the relevance of our proposals, we apply the method of instantiation to two types of network architecture. Then some experiments are conducted and studied.

Keywords : formal verification by temporal model-checking, Ethernet PowerLink (EPL), modelling, abstraction, Uppaal, availability