MODELISATION DES RESEAUX EN ALTARICA 3.0

MODELING NETWORK SYSTEMS WITH ALTARICA 3.0

Tatiana PROSVIRNOVA et Antoine RAUZY
LIX, Ecole Polytechnique
Bât. Alan Turing
91120 Palaiseau
+33 (0)1 77 57 80 91
prosvirnova@lix.polytechnique.fr
rauzy@lix.polytechnique.fr

Frédéric MILCENT
DCNS
Rue du Pont Neuf – CS 81030
16600 RUELLE SUR TOUVRE
+33 (0)5 45 24 30 00
+33 (0)5 45 24 33 33 (fax)
frederic.milcent@dcnsgroup.com

Résumé

L'évaluation de la fiabilité des systèmes comportant des boucles de rétroaction instantanées, tels que les réseaux de communication, posent des problèmes spécifiques. La nouvelle version du langage AltaRica, AltaRica 3.0, permet de les traiter de manière élégante. Dans cette communication, nous proposons une méthodologie de modélisation pour ce type de systèmes. Nous illustrons notre démarche sur un exemple non-trivial. Nous présentons les résultats obtenus sur cet exemple via la génération automatique d'arbres de défaillance.

Summary

The assessment of the reliability of systems with instant feedback loops such as communication networks turn out to be especially hard. The new version of the language AltaRica 3.0 makes it possible to handle this kind of systems in an elegant way. In this communication, we propose a modeling methodology for networks. We illustrate our approach by means of a non-trivial example. We present the results we obtained on this example by means automated Fault Tree generation.

Introduction

Dans cet article, nous considérons le problème de l'évaluation de la fiabilité des réseaux de communications. Ce problème peut être énoncé comme suit. Soit un réseau de communications composé des trois types d'éléments suivants :

- des sources d'information (par exemple, des postes de travail) ;
- des relais (par exemple, des switches) ;
- des cibles (par exemple, des unités de traitement).

Les données sont acheminées depuis les sources vers les cibles à travers les relais connectés entre eux. Elles peuvent circuler dans les deux sens entre deux relais. Tous les éléments du réseau peuvent tomber en panne. Pour plus de simplicité, nous supposons que les événements de panne suivent tous une loi exponentielle (chaque unité ayant potentiellement son propre taux de défaillance). Le système est considéré en panne si plus aucune cible n'est à même de traiter les données émises par les sources, soit qu'elle soit en panne, soit qu'elle ne reçoive plus de données.

La difficulté de modélisation de ce type de système réside dans le fait que les données peuvent circuler dans tous les sens entre les relais et que le flux d'information peut changer de sens suivant l'état des différents composants du réseau. Plus formellement, le graphe non-orienté sous-jacent comporte des cycles. Par conséquent, les systèmes réseaux font partie des systèmes, dit bouclés.

La construction manuelle d'un arbre de défaillance pour un tel réseau est difficile car il faut considérer toutes les combinaisons de pannes possibles et analyser la propagation de l'information dans le réseau afin de déterminer si cette combinaison amène à l'événement redouté.

Dans ce contexte, l'utilisation des langages de modélisation de haut-niveau comme AltaRica présente un grand avantage.

Créé à la fin des années 90, le langage AltaRica (Boiteau, Dutuit, Rauzy, & Signoret, 2006) est un langage de modélisation de haut-niveau pour la sûreté de fonctionnement. Il permet de décrire des modèles à un haut niveau d'abstraction, très proche de l'architecture fonctionnelle ou logique des systèmes. Il possède un certain nombre de bonnes propriétés :

- Basé sur les événements, AltaRica est dédié à la modélisation du comportement non-déterministe ;
- C'est un langage compositionnel : il est possible de représenter le modèle du système sous forme de hiérarchies de composants réutilisables ;
- Il permet de représenter la propagation des flux à travers le système grâce aux variables de flux et à l'assertion;

- C'est un langage formel ce qui a permis de développer des algorithmes de traitement efficaces (compilation vers les arbres de défaillance, génération de séquences critiques, simulation stochastique, etc.).

Il est aussi possible d'associer aux modèles AltaRica des représentations graphiques, les rendant très proches d'un « Process & Instrumentation Diagram ». Le langage AltaRica est utilisé par plusieurs ateliers de sûreté de fonctionnement : Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys) et Safety Designer (Dassault Systèmes). De nombreuses expériences industrielles réussies ont été menées avec le langage AltaRica (voir par exemple (Bieber, et al., 2008), (Bernard, Aubert, Bieber, Merlini, & Metge, 2007)). Cependant, AltaRica Data-Flow, la version du langage utilisée dans les ateliers précités, impose une contrainte forte sur les assertions décrivant la propagation des informations entre les composants. Cette contrainte rend la modélisation des systèmes bouclés difficiles.

La nouvelle version du langage AltaRica 3.0 (Prosvirnova, et al., 2013), basée sur le formalisme des Systèmes de Transitions Gardées, permet de modéliser les systèmes bouclés. Elle autorise aussi la création de composants acausaux, autrement dit de composants avec des flux bidirectionnels. Cette dernière propriété facilite grandement la modélisation des réseaux.

Le projet AltaRica 3.0 vise à développer la nouvelle version du langage AltaRica, ainsi que des outils de traitement associés :

- Compilateurs vers les arbres de défaillance et séquences critiques (Prosvirnova & Rauzy, 2013) ;
- Générateur de graphes de Markov à profondeur limité (Brameret, Rauzy, & Roussel, 2013) ;
- Simulateur stochastique (Batteux & Rauzy, 2013), etc.

Dans cette communication, nous proposons un schéma de modélisation des réseaux de communication en AltaRica 3.0 et nous illustrons notre présentation à l'aide d'un exemple « fil rouge ». Deux points importants doivent être soulignés à propos de ce schéma :

- D'une part, il s'agit avant tout un exemple dont les analystes peuvent s'inspirer et il n'est exclusif d'aucune construction AltaRica 3.0. Autrement dit, il peut être intégré dans des composants de plus haut niveau ou intégrer des composants différents de ceux présentés ici. La notion de schéma de modélisation est à rapprocher des schémas de conception que l'on trouve en ingénierie logicielle (Gamma, Helm, Johnson, & Vlissides, 1994)
- D'autre part, les modèles présentés ici sont susceptibles d'être traités avec les outils AltaRica 3.0 ordinaires (mentionnés ci-dessus). Aucun algorithme spécifique ni nécessaire, ni d'ailleurs utile. Dans le cadre de cet article, nous utilisons le générateur d'arbres de défaillance parce que le réseau que nous traitons ne présente pas de dépendance temporelle entre les pannes.

Ces deux points font l'originalité et tout l'intérêt de l'approche présentée ici, par rapport à des approches s'appuyant sur des langages spécifiques comme les réseaux de fiabilité (Colbourn, 1987) ou les digraphs (Madre, Coudert, Fraïssé, & Bouissou, 1994).

Le reste de l'article est organisé comme suit : nous présentons dans la prochaine section un exemple de réseau que nous utiliserons comme fil rouge pour introduire les différents concepts du langage AltaRica 3.0. Nous discutons des difficultés de modélisation de ce type de systèmes. Nous proposons un schéma de modélisation pour les réseaux de communication. Finalement nous présentons quelques résultats qualitatifs et quantitatifs obtenus par génération automatique d'arbres de défaillance.

Cas d'étude

Le cas d'étude suivant a été proposé par DCNS. Le réseau, représenté Figure 1, est composé de :

- Six postes de travail « Workstation 1 », « Worsktation 2 », ..., « Workstation 6 » produisant les données ;
- Deux unités de traitement « Processing unit 1 » et « Processing unit 2 » en charge de recevoir et analyser les données ;
- Huit relais « Switch 1 », « Switch 2 », ..., « Switch 8 » en charge de recevoir les données des postes de travail et/ou des autres relais et de les transmettre vers les autres relais et/ou vers les unités de traitement.

Nous considérons que tous les composants peuvent tomber en panne et que les événements de panne suivent une loi exponentielle avec les paramètres lambda donnés (voir Figure 1).

Le système est considéré comme défaillant si aucune des deux unités de traitement ne transmet les données aux installations.

Les connections entre les relais sont bidirectionnelles : chaque relais reçoit les données de tous ses voisins et transmet les données reçues vers tous ses voisins. Le sens de circulation de l'information peut changer en cours de mission, suivant l'état des composants. Par exemple si elle peut circuler du poste de travail 1 aux unités de traitement 1 et 2 via les relais 1, 8, 7 et 6 (dans l'ordre) ou via les relais 1, 2, 3, 4, 5 et 6 (dans l'ordre).

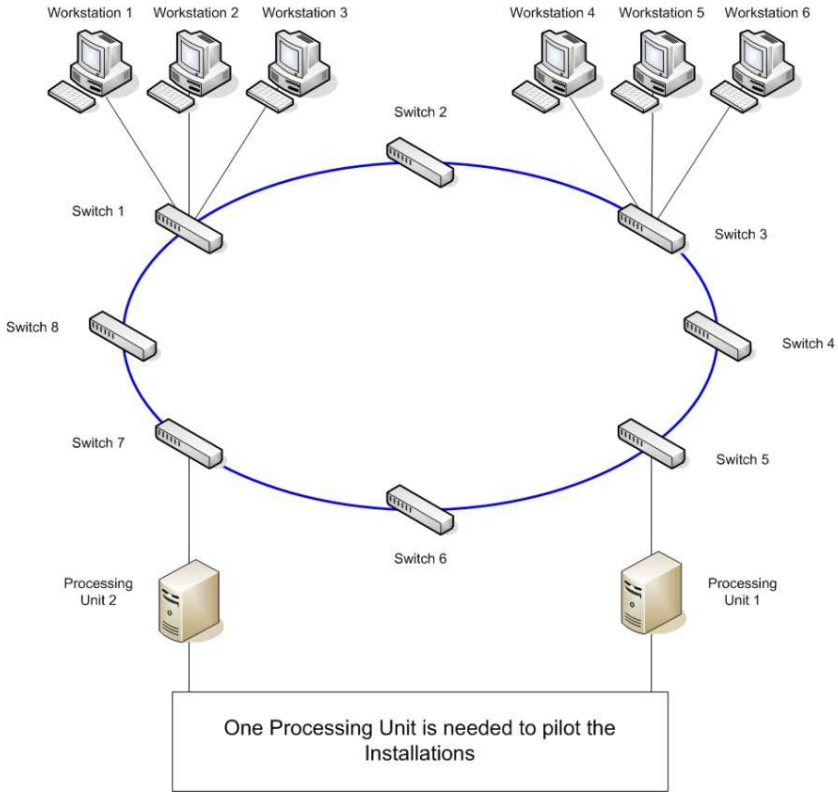

Figure 1. Un réseau de communication

Composant	**Loi**	**Paramètres**
Poste de travail	Exponentielle	lambda = 10^{-5} h^{-1}
Relais	Exponentielle	lambda = 10^{-4} h^{-1}
Unité de traitement	Exponentielle	lambda = 10^{-4} h^{-1}

Table 1. Paramètres des composants du réseau

Cet exemple est le fil rouge de notre article : il sera utilisé pour illustrer les différents concepts du langage AltaRica 3.0 et l'utilisation du schéma de modélisation des réseaux proposé.

Un système similaire a été modélisé en AltaRica Data-Flow dans (Riera, Milcent, J.Parisot, & Clement, 2012). Le système réseau, proposé par DCNS, a été réalisé dans l'atelier Simfia. Plusieurs difficultés de modélisation ont été rencontrées :

- Premièrement, le système étant bouclé par nature, les liaisons au niveau des Switches 1 et 3 ont été "coupées" pour respecter la contrainte Data-Flow.
- Ensuite, l'unidirectionnalité des flux a obligé à doubler les connexions pour pouvoir représenter la transmission des données dans les deux sens.
- Enfin, le fait qu'il y ait deux points de connexion pour les postes de travail a obligé, de nouveau, à doubler les connexions : deux flux pour transmettre les données issues des postes de travail 1, 2 et 3 et encore deux flux pour les postes de travail 4, 5 et 6.

Le langage AltaRica 3.0

Le langage AltaRica est un langage événementiel : on suppose le système étudié ne change d'état que lorsqu'un événement se produit. L'état du système est défini par la valeur de variables. Les changements d'états sont décrits par des transitions étiquetées par des événements. Des délais stochastiques ou déterministes peuvent être associés aux événements pour créer

les modèles temporisés ou stochastiques. Les modèles de composants peuvent être assemblés en hiérarchies, connectés entre eux et leurs événements peuvent être synchronisés.

AltaRica 3.0 est la troisième version du langage AltaRica. Les trois versions diffèrent dans la manière de calculer les valeurs des variables dans les assertions après chaque tir de transition. Dans la première version du langage, les variables étaient calculées par résolution de systèmes de contraintes (Point & Rauzy, 1999). Ce mécanisme très puissant ne s'est pas avéré assez efficace pour des applications de taille industrielle. C'est pourquoi la deuxième version, AltaRica Data-Flow, a été créée. En AltaRica Data-Flow, les variables dans les assertions sont calculées dans un ordre fixé à l'avance. Cet ordre est déterminé à la compilation du modèle, ce qui impose une contrainte forte sur les assertions, et ne permet pas de représenter les systèmes bouclés. La nouvelle version du langage, AltaRica 3.0, améliore les précédentes selon deux axes :

- Un nouveau paradigme de structuration des modèles a été adopté : la modélisation orientée prototypes. De nouvelles constructions structurelles ont été introduites. Voir (Prosvirnova & Rauzy, 2014) pour plus de détails à ce sujet.
- Le modèle d'exécution est désormais basé sur le formalisme des Systèmes de Transitions Gardées (GTS) (Prosvirnova, et al., 2013), ce qui permet de modéliser les systèmes bouclés et les composants acausaux, c'est-à-dire les composants pour lesquels les entrées-sorties sont déterminées dynamiquement à l'exécution du modèle.

La syntaxe a aussi été revue de fond en comble de façon à se rapprocher de celle de Modelica (Fritzson, 2003).

Modélisation d'un composant non réparable

Le code AltaRica 3.0 modélisant un composant non-réparable est donné Figure 2.

```
domain ComponentState {WORKING, FAILED}

class NonRepairableComponent
  ComponentState self (init = WORKING);
  parameter Real lambda = 0.0001;
  event failure (delay = exponential(lambda));
transition
  failure: self==WORKING -> self := FAILED;
end
```

Figure 2. Code AltaRica 3.0 pour un composant non-réparable

La classe "NonRepairableComponent" décrit, comme son nom l'indique, un composant non-réparable générique. L'état de ce composant est représenté par la variable « self » qui prend sa valeur dans le domaine « ComponentState ». Sa valeur initiale est donnée par l'attribut « init » et est égale à WORKING. Le composant change de l'état de « WORKING » à « FAILED » quand l'événement « failure » se produit. On associe à cet événement une loi de probabilité exponentielle avec un paramètre « lambda » via l'attribut « delay ».

La classe « RepairableComponent » (Figure 3) modélise un composant réparable générique. Le composant change d'état de « WORKING » à « FAILED » quand l'événement « failure » se produit et de « FAILED » à « WORKING » quand l'événement « repair » se produit. Comme pour l'événement « failure », on associe à l'événement « repair » une loi de probabilité exponentielle de paramètre « mu ».

La classe « RepairableComponent » est en fait la classe « NonRepairableComponent » dans laquelle un nouveau paramètre « mu », un nouvel événement « repair » et la transition correspondante ont été rajoutés. Comme d'autres langages orientés objets, le langage AltaRica 3.0 introduit la notion d'héritage de classes. Il est possible de construire la classe « RepairableComponent » à partir de la classe « NonRepairableComponent », ainsi qu'il est montré Figure 4. La classe « RepairableComponent » est maintenant dérivée de la classe « NonRepairableComponent » : elle hérite donc de toutes les variables, de tous les événements, de tous les paramètres et de toutes les transitions de la classe « NonRepairableComponent ». Les deux définitions de la classe « RepairableComponent » sont équivalentes.

```
class RepairableComponent
  ComponentState self (init = WORKING);
  parameter Real lambda = 0.0001;
  parameter Real mu = 0.01;
  event failure (delay = exponential(lambda));
  event repair (delay = exponential(mu));
transition
  failure: self==WORKING -> self := FAILED;
  repair: self==FAILED -> self := WORKING;
end
```

Figure 3. Code AltaRica 3.0 pour un composant réparable

```
class RepairableComponent
  extends NonRepairableComponent;
  parameter Real mu = 0.01;
  event repair (delay = exponential(mu));
transition
  repair: self==FAILED -> self := WORKING;
end
```

Figure 4. Code AltaRica 3.0 pour un composant réparable utilisant l'héritage

Dans le code AltaRica 3.0 donné Figure 5, la classe « StandbyComponent » modélise un composant en réserve (en redondance froide). Le composant peut être dans trois états différents : « STANDBY », « WORKING » et « FAILED ». A l'état initial, il est dans l'état « STANDBY ». En plus de la variable d'état « self », la classe possède une variable Booléenne « demanded », dite variable de flux. AltaRica 3.0 introduit deux types de variables : les variables d'état, repérées par l'attribut "init", et les variables de flux, repérées par l'attribut « reset ». Les variables d'état sont utilisées pour modéliser l'état des composants. Leur valeur est modifiée par les actions des transitions. Les variables de flux sont utilisées pour représenter les flux de données (de matière, d'électricité, etc.) qui circulent entre les composants. Leur valeur est mise à jour après chaque tir de transition par l'assertion. Dans notre exemple, la variable « demanded » est utilisée pour modéliser la commande : quand sa valeur passe à vrai, le composant doit démarrer. Cette opération est immédiate (son délai est nul), mais une défaillance à la sollicitation peut se produire avec une probabilité « gamma » (événement "failureOnDemand").

```
domain StandbyComponentState {STANDBY, WORKING, FAILED}

class StandbyComponent
  StandbyComponentState self (init = STANDBY);
  Boolean demanded(reset = FALSE);
  parameter Real lambda = 0.0001;
  parameter Real gamma = 0.02;
  parameter Real mu = 0.01;
  event failure (delay = exponential(lambda));
  event repair (delay = exponential(mu));
  event start (delay = 0, expectation = 1 - gamma);
  event failureOnDemand (delay = 0, expectation = gamma);
  event stop (delay = 0);
transition
  failure: self==WORKING -> self := FAILED;
  repair: self==FAILED -> self := STANDBY;
  start: self==STANDBY and demanded -> self:=WORKING;
  failureOnDemand: self==STANDBY and demanded -> self:=FAILED;
  stop: self==WORKING and not demanded -> self:= STANDBY;
end
```

Figure 5. Code AltaRica 3.0 pour un composant en redondance froide

D'autres schémas de modélisation des composants de base d'un système peuvent être définis. Dans le cadre de cet article, nous considérons que les composants sont non-réparables. Nous n'utiliserons donc que la classe « NonRepairableComponent » pour modéliser les composants des réseaux.

Schéma de modélisation pour les réseaux de communication

Dans cette section, nous proposons un schéma de modélisation pour les réseaux de communication, au sens défini dans l'introduction. Un tel réseau est donc composé de trois types d'éléments :

- des sources qui produisent les données ;
- des cibles qui reçoivent et analysent les données ;
- des relais ou switches qui reçoivent les données des sources et/ou d'autres relais et les transmettent aux cibles et/ou aux autres relais.

Tous les composants peuvent être défaillants.

Modélisations des composants

Les sources sont modélisées par la classe « Source » donnée Figure 6. Cette classe hérite de la classe « NonRepairableComponent ». La variable de flux Booléenne « outFlow » représente le flux de données produit par la source. Sa valeur par défaut est faux. Si la source est en marche, elle produit des données (la variable « outFlow » est égale à « true »). Si la source est en panne, elle ne produit pas de données (la variable « outFlow » prend la valeur « false »).

```
class Source
  extends NonRepairableComponent;
  Boolean outFlow(reset = FALSE);
assertion
  outFlow := self==WORKING;
end
```

Figure 6. Code AltaRica 3.0 pour les sources.

Les cibles sont représentées par la classe « Target » donnée Figure 7. Cette classe ressemble à la classe « Source » mais elle contient deux variables de flux : la variable « inFlow » représente le flux de données reçu par la cible et la variable « outFlow » le flux de données traité par la cible et envoyé vers les installations. Quand la cible est opérationnelle, elle reçoit les données en entrée, les traite et envoie le résultat en sortie (la valeur de la variable « outFlow » est égale à la valeur de « inFlow »). Si la cible est en panne, elle ne peut pas traiter les données en entrée et ne renvoie aucune donnée (la variable « outFlow » prend la valeur « false »).

```
class Target
  extends NonRepairableComponent;
  Boolean inFlow, outFlow(reset = FALSE);
assertion
  outFlow := self==WORKING and inFlow;
end
```

Figure 7. Code AltaRica 3.0 pour les cibles.

Les composants définis précédemment, « Source » et « Target », sont des composants dit data-flow (le flux de données est orienté). En revanche les relais sont des composants acausaux, c'est-à-dire des composants pour lesquels les entrées et les sorties ne sont pas définies à la compilation mais sont calculées dynamiquement à l'exécution du modèle et dépendent de l'état global du système. La classe "Switch", donnée Figure 8, modélise les relais.

```
class Switch
  extends NonRepairableComponent;
  Boolean inFlow, outFlow(reset = FALSE);
  Boolean leftFlow, rightFlow(reset = FALSE);
assertion
  if self==WORKING then leftFlow := rightFlow or inFlow;
  if self==WORKING then rightFlow := inFlow or leftFlow;
  outFlow := if self==WORKING then inFlow or leftFlow or rightFlow else FALSE;
end
```

Figure 8. Code AltaRica 3.0 pour les relais.

La variable de flux « inFlow » représente le flux de données reçu des sources. Si le relais n'est connecté à aucune source, la valeur de cette variable est toujours égale à sa valeur par défaut (« false »). La variable « outFlow » représente le flux de sortie vers les cibles. Notez que le relais peut ne pas être connecté aux cibles. Les variables de flux « leftFlow » et « rightFlow » modélisent les flux acausaux. Si le relais est en marche, alors tous les flux qui le relient à ses voisins sont à « true » si au moins un des flux est à « true ». Si le relais est en panne, alors il n'y a pas de relations entre les variables de flux.

La classe « Switch » suppose qu'un relai est connecté à au plus deux autres relais. Il est bien sûr possible de définir des relais connectables à un nombre quelconque de voisins. Un langage de script peut être utilisé pour générer automatiquement le code AltaRica 3.0 des relais pour un nombre de voisins donné.

Modélisation du réseau

Une fois la bibliothèque des composants définie, nous l'utilisons désormais pour modéliser le réseau présenté au début de cet article. Cette modélisation est en fait extrêmement simple : il suffit de déclarer les composants et de les connecter entre eux ! Le code AltaRica pour le réseau est donné Figure 9.

```
block NetworkSystem
  Source W1, W2, W3, W4, W5, W6 (lambda = 1.0e-5);
  Target P1, P2;
  Switch SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8;
  observer Boolean failed = not P1.outFlow and not P2.outFlow;
assertion
  SW1.inFlow := W1.outFlow or W2.outFlow or W3.outFlow;
  SW3.inFlow := W4.outFlow or W5.outFlow or W6.outFlow;
  SW1.rightFlow :=: SW2.leftFlow;
  SW2.rightFlow :=: SW3.leftFlow;
  SW3.rightFlow :=: SW4.leftFlow;
  SW4.rightFlow :=: SW5.leftFlow;
  SW5.rightFlow :=: SW6.leftFlow;
  SW6.rightFlow :=: SW7.leftFlow;
  SW7.rightFlow :=: SW8.leftFlow;
  SW8.rightFlow :=: SW1.leftFlow;
  P1.inFlow := SW5.outFlow;
  P2.inFlow := SW7.outFlow;
end
```

Figure 9. Code AltaRica 3.0 pour le réseau de communication

Le bloc « NetworkSystem » est un modèle hiérarchique : il contient six instances de la classe « Source » représentant les postes de travail, deux instances de la classe « Target », représentant les unités de traitement, et huit instances de la classe « Switch » modélisant les relais.

La valeur du paramètre « lambda » pour les postes de travail est modifiée à l'instanciation et est désormais égale à 10^{-5}. La valeur du paramètre « lambda » pour les autres composants reste égale à 10^{-4} comme défini dans la classe « NonRepairableComponent ». Les valeurs des attributs et des paramètres peuvent être modifiées au moment de l'instanciation des classes.

Le code de la Figure 9 définit un observateur booléen « failed » : sa valeur est « true » quand le système est en panne, c'est-à-dire quand aucune des unités de traitement n'est capable d'envoyer les données. Les observateurs sont comme les variables de flux mais ils ne peuvent pas être utilisés dans les transitions ou dans les assertions. Ce sont des quantités à observer par les outils de traitement.

L'assertion représente les connections entre les composants. L'opérateur « :=: » définit les connections bidirectionnelles : les données peuvent circuler dans les deux sens entre les relais. La direction de propagation des flux est déterminée à l'exécution parce qu'elle dépend de l'état global du système.

Illustrons le mécanisme de calcul des variables de flux. Supposons d'abord que les postes de travail W4, W5 et W6 sont en panne, ainsi que le relai SW4. Dans cette configuration, les données sont propagées depuis les postes de travail W1, W2 et W3 vers le relai SW1, ensuite vers l'unité de traitement P2 via les switches SW8, SW7 et SW5 dans l'ordre, et enfin vers l'unité de traitement P1 via les switches SW6 et SW5. Maintenant supposons que le switch SW4 est en marche et le switch SW8 est en panne. Dans cette configuration les données sont propagées depuis les postes de travail vers les unités de traitement via les switches SW1, SW2, SW3, SW4, SW5, SW6, SW7 dans l'ordre. Dans le premier cas le flux circule de SW7 vers SW6 et de SW6 vers SW5. Dans le second cas, il circule dans le sens inverse. Si les deux switches SW4 et SW8 sont en panne alors les données ne peuvent pas être propagées vers les unités de traitement. Les valeurs des variables de flux qui ne peuvent pas être calculées sont remises à leurs valeurs par défaut.

Les réseaux de communication sont bouclés par nature. Aussi les connections entre les relais sont bidirectionnelles. Grâce à la capacité d'AltaRica 3.0 à représenter les systèmes bouclés et les composants acausaux, la modélisation des réseaux est simplifiée : d'une part, il n'est plus utile de doubler les connections pour représenter les flux bidirectionnels ; d'autre part, il n'est plus nécessaire de couper les liaisons entre les relais pour respecter la contrainte data-flow.

Résultats Expérimentaux

Le modèle du réseau de communication a été réalisé dans l'éditeur de texte AltaRica 3.0, basé sur Xtext d'Eclipse (voir Figure 10). Cet éditeur possède les fonctionnalités suivantes :

- Structuration des modèles dans des projets,
- Coloration syntaxique,
- Vérification lexicale et syntaxique à la volée,
- Vue arborescente des modèles.

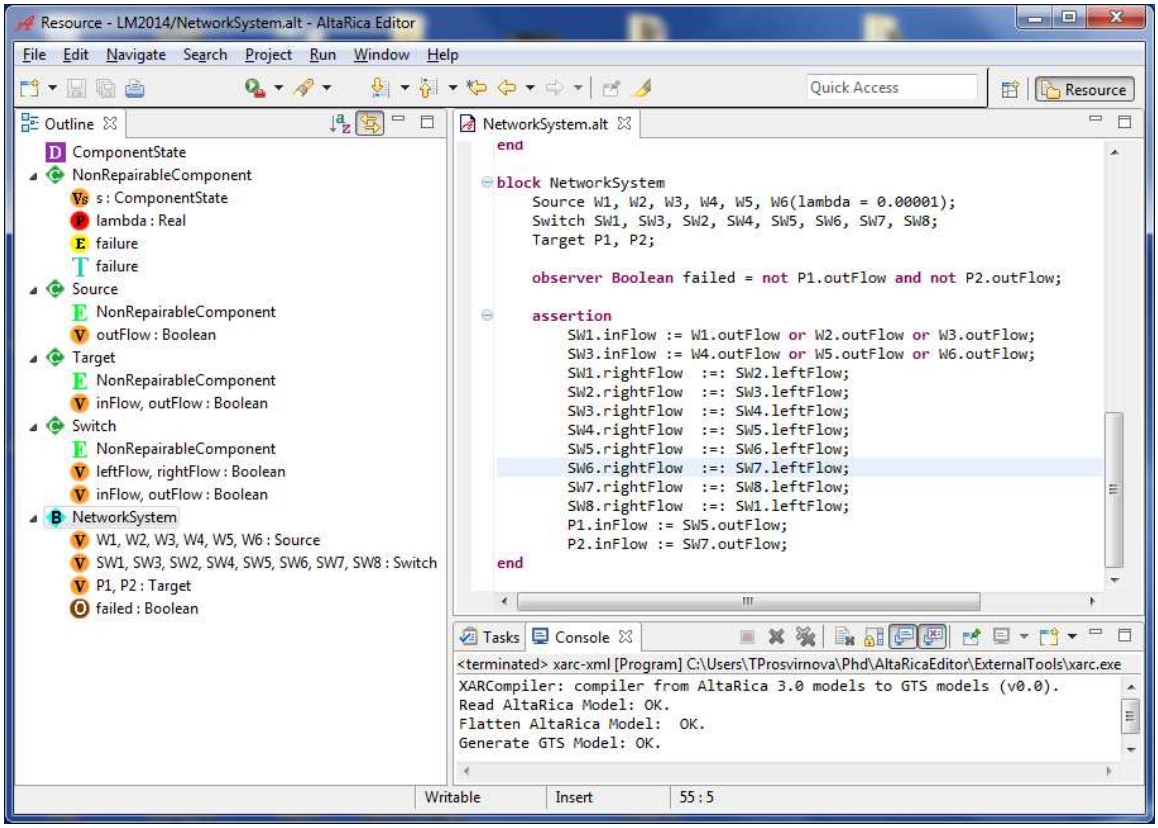

Figure 10. Atelier de création des modèles AltaRica 3.0

Pour assurer l'efficacité de traitement, tout modèle AltaRica 3.0 est d'abord compilé vers les Systèmes de Transitions Gardées (GTS). Dans le cadre du projet AltaRica 3.0, un compilateur de modèles AltaRica 3.0 vers les GTS a été développé. Il est intégré à l'éditeur AltaRica 3.0. Le modèle GTS compilé peut ensuite être analysé avec différents outils de traitement.

Dans le cadre de cette étude, nous présentons l'analyse du modèle par génération automatique d'arbres de défaillance. Le GTS doit ensuite être compilé vers un arbre de défaillance (ADD). Cet arbre de défaillance est imprimé au format Open PSA (Hibti, Friedlhuber, & Rauzy, 2012) et peut ensuite être analysé par l'outil XFTA (Rauzy, 2012) pour déterminer les coupes minimales et la probabilité de panne du système. L'algorithme de compilation s'appuie sur une technique de partitionnement du modèle pour augmenter l'efficacité de traitement (Prosvirnova & Rauzy, 2013).

Le modèle GTS contient 58 variables et 16 événements. Le graphe d'accessibilité complet contiendrait 65536 états et 524288 transitions. Le modèle partitionné est composé de 16 partitions, le graphe d'accessibilité de chaque partition contient seulement 2 états et 1 transition. A titre indicatif, le temps d'exécution du programme avec partitionnement est de l'ordre de 0.25 secondes, sans partitionnement de l'ordre de 61 secondes.

Les résultats obtenus sont présentés dans la Table 2.

Conclusion

Dans cet article, nous avons présenté la nouvelle version du langage AltaRica, AltaRica 3.0. Cette nouvelle version permet entre autres de modéliser les systèmes bouclés et de représenter les composants acausaux. Nous avons développé une bibliothèque de composants pour la modélisation des systèmes réseaux. Nous avons ensuite illustré l'utilisation de cette bibliothèque sur un cas-test proposé par DCNS.

Les réseaux de communication sont bouclés par nature. Aussi les connections entre les composants sont bidirectionnelles. Grâce à la capacité d'AltaRica 3.0 à représenter les systèmes bouclés et les composants acausaux, la modélisation des réseaux est simplifiée : d'une part, il n'est plus utile de doubler les connections pour représenter les flux bidirectionnels ; d'autre part, il n'est plus nécessaire de couper les liaisons entre les composants pour respecter la contrainte data-flow.

Rang	Ordre	Probabilité	Contribution	Coupes minimales					
1	2	9,0559E-03	8,0771E-02	P1.failure	P2.failure				
2	2	9,0559E-03	8,0771E-02	P1.failure	SW7.failure				
3	2	9,0559E-03	8,0771E-02	P2.failure	SW5.failure				
4	2	9,0559E-03	8,0771E-02	SW5.failure	SW7.failure				
5	2	9,0559E-03	8,0771E-02	SW4.failure	SW7.failure				
6	2	9,0559E-03	8,0771E-02	SW3.failure	SW7.failure				
7	2	9,0559E-03	8,0771E-02	SW5.failure	SW8.failure				
8	2	9,0559E-03	8,0771E-02	SW1.failure	SW5.failure				
9	2	9,0559E-03	8,0771E-02	SW3.failure	SW8.failure				
10	2	9,0559E-03	8,0771E-02	SW4.failure	SW8.failure				
11	2	9,0559E-03	8,0771E-02	SW1.failure	SW4.failure				
12	2	9,0559E-03	8,0771E-02	SW1.failure	SW3.failure				
13	3	8,6178E-04	7,6864E-03	P1.failure	SW6.failure	SW8.failure			
14	3	8,6178E-04	7,6864E-03	P1.failure	SW1.failure	SW6.failure			
15	3	8,6178E-04	7,6864E-03	P2.failure	SW4.failure	SW6.failure			
16	3	8,6178E-04	7,6864E-03	P2.failure	SW3.failure	SW6.failure			
17	4	9,3747E-08	8,3614E-07	SW3.failure	W1.failure	W2.failure	W3.failure		
18	4	9,3747E-08	8,3614E-07	SW1.failure	W4.failure	W5.failure	W6.failure		
19	5	8,9212E-09	7,9570E-08	SW2.failure	SW7.failure	W4.failure	W5.failure	W6.failure	
20	5	8,9212E-09	7,9570E-08	SW2.failure	SW5.failure	W1.failure	W2.failure	W3.failure	
21	5	8,9212E-09	7,9570E-08	SW2.failure	SW8.failure	W4.failure	W5.failure	W6.failure	
22	5	8,9212E-09	7,9570E-08	SW2.failure	SW4.failure	W1.failure	W2.failure	W3.failure	
23	6	8,4897E-10	7,5720E-09	P1.failure	SW2.failure	SW6.failure	W1.failure	W2.failure	W3.failure
24	6	8,4897E-10	7,5720E-09	P2.failure	SW2.failure	SW6.failure	W4.failure	W5.failure	W6.failure
25	6	9,7047E-13	8,6558E-12	W1.failure	W2.failure	W3.failure	W4.failure	W5.failure	W6.failure

Table 2. Coupes minimales et leurs probabilités pour le système réseau

Nous avons analysé le modèle du cas-test par génération automatique d'un arbre de défaillance. L'arbre de défaillance généré a été traité avec l'outil de calcul XFTA pour déterminer les coupes minimales et leurs probabilités. Nous avons obtenu les résultats attendus pour ce système mais avec une modélisation plus simple grâce à l'utilisation des flux bidirectionnels.

Bibliographie

Batteux, M., & Rauzy, A. (2013). Stochastic Simulation of AltaRica 3.0 models. *Proceedings of the European Safety and Reliability Conference, ESREL 2013.* Amsterdam (The Netherlands).

Bernard, R., Aubert, J.-J., Bieber, P., Merlini, C., & Metge, S. (2007). Experiments in model-based safety analysis: flight controls. *Proceedings of IFAC workshop on Dependable Control of Discrete Systems* (pp. 43-48). Cachan (France): Curran Associates, Inc.

Bieber, P., Blanquart, J.-P., Durrieu, G., Lesens, D., Lucotte, J., Tardy, F., et al. (2008). Integration of formal fault analysis in ASSERT: Case studies and lessons learnt. *Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008.* Toulouse (France): SIA (electronic proceedings).

Boiteau, M., Dutuit, Y., Rauzy, A., & Signoret, J.-P. (2006). The AltaRica Data-Flow language in use: Assessment of Production Availability of a MultiStates System. *Reliability Engineering and System Safety , 91*, 747-755.

Brameret, P.-A., Rauzy, A., & Roussel, J.-M. (2013). Preliminary System Safety Analysis with Limited Markov Chain Generation. *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS'2013* (pp. 13--18). York, Great Britain: International Federation of Automatic Control.

Colbourn, C. J. (1987). *The combinatorics of network reliability*. New York: Oxford University Press.

Fritzson, P. (2003). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Hoboken, NJ 07030-5774, USA: Wiley-IEEE Press.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns – Elements of Reusable Object-Oriented Software.* Boston, MA 02116, USA: Addison-Wesley.

Hibti, M., Friedlhuber, T., & Rauzy, A. (2012). Overview of The Open PSA Platform. Dans R. Virolainen (Éd.), *Proceedings of International Joint Conference PSAM'11/ESREL'12.* Helsinki (Finland).

Madre, J.-C., Coudert, O., Fraïssé, H., & Bouissou, M. (1994). Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis. *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94* (pp. 118-123). Annaheim: IEEE.

Point, G., & Rauzy, A. (1999). AltaRica: Constraint automata as a description language. *Journal Européen des Systèmes Automatisés , 33*, 1033-1052.

Prosvirnova, T., & Rauzy, A. (2013). AltaRica 3.0 project: compile Guarded Transition Systems into Fault Trees. *European Safety and Reliability Conference, ESREL 2013.* Amsterdam (The Netherlands).

Prosvirnova, T., & Rauzy, A. (2014). Les constructions structurelles de AltaRica 3.0. *Actes du Congrés Lambda-Mu 19.* Dijon (France).

Prosvirnova, T., Batteux, M., Brameret, P.-A., Cherfi, A., Friedlhuber, T., Roussel, J.-M., et al. (2013). The AltaRica 3.0 project for Model-Based Safety Assessment. *4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2013.* York (Great Britain): IFAC.

Rauzy, A. (2012). XFTA : pour que cent arbres de défaillance fleurissent au printemps. *Actes du Congrès Lambda-Mu 18.* Tours (France).

Riera, D., Milcent, F., J.Parisot, & Clement, E. (2012). Modélisation dynamique en sûreté de fonctionnement : une avancée pour la modélisation des systèmes complexes. Dans J. Barbet (Éd.), *Actes du Congrès Lambda-Mu 18.* Tours (France).