

UN POINT DE VUE SÛRETÉ DE FONCTIONNEMENT POUR LA VÉRIFICATION D'ARCHITECTURES ABSTRAITES

MARIA PIRIQUITO, PIERRE BIEBER, JULIEN BRUNEL, AND DAVID CHEMOUIL

Publié dans *Conférence francophone sur les architectures logicielles*, Montpellier, 2012.

RÉSUMÉ. La vérification de propriétés de sûreté de fonctionnement (SdF) pour les systèmes critiques est un problème primordial. Les solutions existantes imposent généralement une connaissance de l'architecture complète du système. Dans cet article, nous proposons une approche permettant de vérifier efficacement la sûreté de fonctionnement d'architectures abstraites qui sont, éventuellement, partiellement décrites. Nous présentons une méthodologie qui s'appuie sur une vue SdF et une vue architecturale du système. Des propriétés de SdF sont vérifiées au niveau de la vue idoine, et sont postérieurement validées sur l'architecture. Cette approche permet donc de décomposer des vérifications complexes de SdF sur des architectures en deux vérifications plus simples, sur deux vues différentes. Les techniques présentées sont appliquées au système hydraulique d'un avion, utilisé comme cas d'étude.

1. INTRODUCTION

La Sûreté de Fonctionnement s'est développée principalement à cause de l'évolution des systèmes critiques industriels, et se caractérise par l'analyse des défaillances et de leurs conséquences. Cela passe par une analyse exhaustive du fonctionnement du système, ainsi que des exigences que le système doit vérifier.

Les exigences de sûreté de fonctionnement déterminent ce qu'un système doit ou ne doit pas faire pour assurer son bon fonctionnement [Sto99]. Notre travail se place dans le domaine de la *tolérance aux pannes*, *c-a-d* les techniques qui permettent au système de continuer à fonctionner correctement en présence de pannes. Notre but principal est de fournir une réponse rapide et efficace à la vérification de certaines exigences de tolérance aux pannes sur une architecture abstraite et, éventuellement, partiellement décrite. Dans notre approche nous décomposons les vérifications complexes de SdF en deux vérifications plus simples, une au niveau de la vue SdF, et une autre au niveau de la vue architecturale. D'autres approches au problème fournissent des résultats plus proches de la réalité, mais ce sont aussi des méthodes très complexes qui nécessitent une connaissance très approfondie du système (Altarica [APGRoo, Poioo]).

Les Langages de Description d'Architecture (ADL) sont utilisées pour décrire des architectures de système en utilisant les concepts de composants et connecteurs. Dans ce travail, nous proposons un noyau de concepts ADL qui permet une description abstraite de l'architecture d'un système à étudier. Ce langage permet la définition de composants et leur lien par l'intermédiaire de connecteurs prédéfinis. Les détails sur le langage sont donnés dans la Section 2.

La construction du point de vue sûreté de fonctionnement est basé sur la définition de plusieurs vues sûreté de fonctionnement. Chaque vue SdF propose une solution à une exigence différente. C'est l'ensemble de toutes ces vues qui compose le point de vue SdF. La définition de chaque vue est basée sur l'application successive de stratégies de tolérance aux pannes aux composants du système. Les exigences traitées et la méthode de construction des vues sont décrits dans la Section 3.

Pour faire le lien entre le point de vue SdF et la vue architecturale du système nous introduisons la notion de *zone*. Les *zones* mettent en relation les éléments de la vue architecturale et les stratégies de la

vue SdF, et regroupent un ensemble de composants et connecteurs de l'architecture. Chaque application d'une stratégie SdF dans une vue peut donner lieu à une zone, dans le cas où tous les critères d'application de la stratégie sont vérifiés par l'architecture comprise dans la zone. Si toutes les zones créées satisfont tous les critères, il est possible de dire que l'architecture satisfait l'exigence de tolérance aux pannes représentée par la vue SdF correspondante. La méthodologie utilisée est décrite dans la Section 3.4.

Pour mieux expliquer notre méthodologie nous proposons l'application des concepts introduits à un cas d'étude décrivant le système hydraulique d'un avion (Section 2.2).

2. VUE ARCHITECTURALE

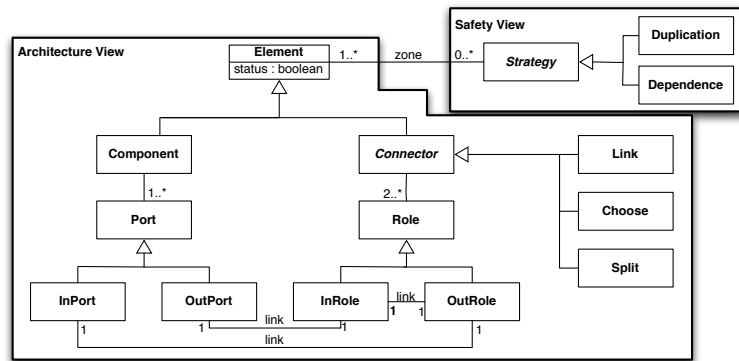


FIG. 1. *Meta-Modèle des Éléments des Différentes Vues de notre Approche*

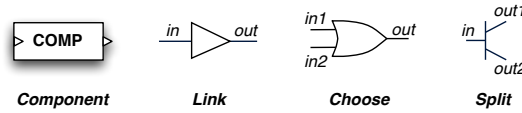


FIG. 2. *Syntaxe des éléments de l'ADL*

Dans cet article, nous introduisons un noyau de concepts présents dans les ADL qui permet la description d'une vue architecturale d'un système. Le meta-modèle qui représente les éléments de cette vue, ainsi que de la vue SdF et le lien entre les deux par les *zones* provenant des stratégies SdF utilisées est présenté dans la Figure 1. En plus de ce qui est représenté dans le meta-modèle, certaines contraintes doivent être vérifiées: (1) Les connecteurs doivent avoir au moins un InRole et un OutRole; (2) une Stratégie est toujours lié par une zone à, au moins, un Composant; (3) un connecteur entre deux composants de la même zone appartient à la zone; et (4) un connecteur entre deux composants de deux zones différentes n'appartiennent pas à la zone.

Nous présentons maintenant la syntaxe du langage suivie de son utilisation en l'utilisant pour décrire le système hydraulique d'un avion.

2.1. Syntaxe et Sémantique des Connecteurs. Notre ADL permet la définition de *composants* et *connecteurs*. Un *Composant* représente une fonction du système à décrire, et contient un nom comme identifiant. Chaque composant doit avoir au moins un *port* d'entrée ou de sortie. Un *connecteur* représente une façon de lier deux composants entre eux. Chaque connecteur doit avoir au moins un *rôle* d'entrée et un de sortie.

Les deux éléments du langage, les composants et les connecteurs, ont un *statut*. Le statut d'un élément représente son intervention dans l'architecture à un certain moment.

Définition 1. *Un composant inactif est un composant qui n'introduit pas de pannes dans le système.*

Définition 2. *Un connecteur inactif est un connecteur qui ne propage pas les pannes dans le système.*

Pour la construction de la vue architecturale du système, tous les composants et connecteurs sont considérés comme étant actifs. L'inactivité d'un élément n'est considérée qu'à la définition d'une vue SdF.

2.1.1. *Composants et Ports.* Les *composants* sont un des concepts de base de notre langage et représentent les fonctions du système. Les *ports* sur un composant représentent des connexions possibles entre ce composant et d'autres composant. Au niveau de la syntaxe, les composants sont des rectangles, les *InPorts* sont les triangles à l'intérieur de la boîte, et les *OutPorts* sont des triangles à l'extérieur de la boîte. Chaque composant a un nom qui permet son identification (Figure 1).

Tout composant actif peut introduire une panne dans le système. Les pannes introduites par les composants seront propagées dans le système par les connecteurs.

Nous présentons de suite les trois connecteurs de base inclus dans le langage, ainsi que leur représentation graphique (Figure 1). Ces connecteurs peuvent se lier entre eux pour créer des connecteurs plus complexes. Nous attirons l'attention sur le fait que la plupart des ADL ne permettent pas cette opération. Nous ne présentons que la sémantique de tolérance aux pannes du langage car c'est celle qui nous intéresse dans cet article.

2.1.2. *Les Connecteurs Link, Choose et Split.* La syntaxe des connecteurs est représentée dans la Figure 2. Les liens entre éléments du langage (ports et rôles) sont symboliques et représentés par une ligne qui lie un port à un rôle, ou un rôle à un rôle.

Le connecteur *Link* sert à définir une connexion standard entre deux composants. Il contient un InRole *in* et un OutRole *out*, et les données reçues dans *in* sont renvoyées vers *out* sans introduire de modifications. Si le connecteur est actif, alors toute panne rentrante en *in* est transmise via *out*.

Le connecteur *Choose* contient deux InRoles *in1* et *in2* et un seul OutRole *out*. Son comportement est similaire à celui du connecteur *Link*, mais en prenant ses données soit de *in1* soit de *in2*. Ce choix est non-déterministe si les deux entrées sont disponibles. Si une seule est disponible, ce sera sa valeur qui sera transmise vers *out*. Dans le cas où le connecteur est actif, alors, si une panne existe dans les deux entrées *in1* et *in2* simultanément, elle sera transmise à la sortie *out*. Si une seule panne existe, elle ne sera pas propagée.

Le connecteur *Split* a un seul InRole *in* et deux OutRoles *out1* et *out2*, et les données entrantes, si existantes, devront être transmises vers *out1* et *out2*. En cas de panne dans l'entrée *in* d'un connecteur *Split* actif, alors celle-ci sera transmise aux deux sorties *out1* et *out2*.

2.2. **Le Système Hydraulique d'un Avion.** Pour aider à la compréhension de notre langage, nous allons l'appliquer à un exemple basé sur le système hydraulique d'un avion. Dans cet exemple les composants utilisés sont d'une part physiques et d'autre part logiques. Chaque composant physique contient un contrôleur logique qui détermine son état: marche, ne marche pas,...

Le système utilisé en tant qu'exemple dans cette Section est basé sur l'exemple décrit en [Sago8], présentant un système hydraulique d'un avion A320. Pour simplifier la présentation de notre travail nous utilisons une version simplifiée de l'exemple. Le système hydraulique que nous présentons est composé par deux lignes de distribution, *vert*, identifiée par l'index *g*, et *jaune*, identifiée par l'index *y*. Les composants *DISTg* et *DISTy* représentent la connexion entre les lignes de distribution et le reste du système (pas représenté dans cet exemple). Pour fournir de l'énergie il y a deux moteurs de l'avion, *ENG1* et *ENG2*, ainsi qu'un générateur électrique *ELEC* pour alimenter la ligne de distribution *jaune*. Chaque ligne de distribution est constituée d'un réservoir, *RSVg* et *RSVy*, connecté aux pompes *EDPg* et *EDPy*

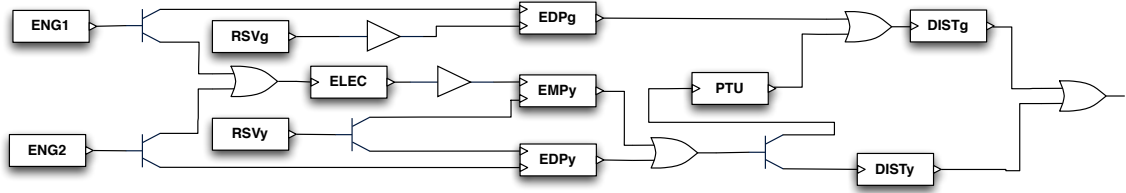


FIG. 3. Architecture Fonctionnelle du Système Hydraulique d'un Avion

respectivement. En cas de défaillance au niveau de la pompe $EDPg$, il y a un autre mécanisme de secours, le PTU qui permet d'alimenter le système de distribution *vert* en utilisant une des pompes du système *jaune*.

En utilisant notre ADL pour définir le système, on obtient l'architecture de la Figure 3. Cette architecture sera utilisée comme base dans les sections suivantes pour appliquer la définition des vues SdF, et pour effectuer la vérification des exigences de SdF sur cette même architecture.

3. VUE SÛRETÉ DE FONCTIONNEMENT POUR LES ARCHITECTURES

Dans cet article, nous nous focalisons sur la tolérance aux pannes dont le développement est essentiellement basé sur la redondance, *c-a-d*, le système contient au moins deux composants fournissant le même service. Dans ce cas, si l'un des composants tombe en panne les autres composants sont toujours aptes à fournir le service. Le système hydraulique décrit dans la figure 3 contient plusieurs composants redondants : les lignes de distribution $DISTy$ et $DISTg$, les pompes $EDPg$, $EMPy$ et $EDPy$ ou les moteurs de $ENG1$ et $ENG2$.

Le niveau de redondance présent dans l'architecture d'un système répond à des exigences de tolérances aux pannes. Pour des raisons de coûts, d'encombrement ou de masse il n'est pas souhaitable de dupliquer tous les composants d'un système. Aussi, il est possible que la panne d'un composant impacte plusieurs composants redondants. Ceci peut avoir pour effet d'invalider des exigences de SdF. Dans le système hydraulique, le réservoir $RSVy$ n'est pas dupliqué, il est utilisé par les pompes $EMPy$ et $EDPy$. Par conséquent, lorsque le réservoir ne fournit plus de fluide hydraulique, les deux pompes $EMPy$ et $EDPy$ ne peuvent plus fournir de fluide sous pression. Dans ce cas, c'est le système hydraulique *vert* qui utilise des composants indépendants de $RSVy$ comme un réservoir $RSVg$, une pompe $EDPg$ et une ligne de distribution $DISTg$ qui fournira la puissance hydraulique.

3.1. Exigences de SdF. Les exigences de SdF ne contraignent pas ce que le système fait (par exemple, fournir de la puissance hydraulique) mais imposent des conditions sur la façon dont c'est fait (par exemple, fournir de la puissance hydraulique lorsque le moteur $ENG1$ est en panne). Ces exigences sont issues de l'analyse des risques qui évalue l'impact de la perte d'un service sur la vie des passagers, sur l'intégrité de l'avion et sur la charge de travail des pilotes. La gravité de l'impact détermine le niveau de tolérance aux pannes requis.

Plusieurs types d'exigences de tolérance aux pannes existent. Nous traitons des exigences permettant de décrire les combinaisons de pannes qu'un système (ou qu'une partie du système) doit tolérer. Ces combinaisons de pannes sont habituellement caractérisées par un nombre de pannes (pannes simples, doubles ou triples). Elles peuvent également décrire un ensemble de composants susceptible de tomber en panne.

Dans notre exemple, lorsque le système hydraulique n'est plus capable de fournir de la pression hydraulique, l'avion n'est plus gouvernable. Par conséquent, l'impact de la perte totale du système hydraulique est très grave. L'architecture décrite dans la figure 3 représente une portion de système

hydraulique dont la perte aurait un impact moindre. Nous considérons dans la suite que ce système doit satisfaire l'exigence 1: Le système hydraulique vert doit tolérer la panne d'un des deux moteurs.

3.2. Les Stratégies de Tolérance aux Pannes. Pour vérifier les exigences de SdF présentées, nous construisons une vue SdF du système. Cette vue va être basée dans la décomposition du système en utilisant des stratégies de tolérance aux fautes introduites en [Bruo9] et adaptées à notre travail: la *Dépendance* et la *Duplication*. Chaque stratégie comprend des hypothèses sur son applicabilité, ainsi que des propriétés garanties. La vérification des hypothèses est effectuée en analysant l'architecture du système (Section 3.4).

Définition 3. Soit c un composant (ou ensemble de composants) du système. Alors la fonction $loss(c)$ représente le nombre minimal de pannes nécessaires pour perdre c .

Définition 4. Soit c un composant (ou ensemble de composants). Si toute autre information de tolérance aux pannes est inexistante, alors:

$$loss(c) = \begin{cases} 1 & \text{si } c \text{ est actif} \\ 0 & \text{si } c \text{ est inactif} \end{cases}$$

3.2.1. La Stratégie Dépendance. Cette stratégie permet la représentation de la dépendance entre plusieurs fonctions du système. L'utilisation de cette stratégie n'apporte rien dans l'amélioration de la tolérance aux pannes. Son utilité est visible au niveau de la composition de stratégies pour créer une vue SdF.

Définition 5. Soient c_1, \dots, c_n plusieurs composants (ou ensemble de composants) du système. Alors une dépendance entre c_1, \dots, c_n est définie par $dep(c_1, \dots, c_n)$.

Hypothèse 1. Soit C l'ensemble des composants c_1, \dots, c_n en dépendance. Alors il existe un connecteur liant chaque composant $c \in C$ à un composant $c' \in C \setminus \{c\}$.

Cette hypothèse garantit que les différentes entités ne sont pas isolées, et que la perte d'une d'entre elles va déclencher la perte de l'ensemble.

Propriété Garantie 1. La tolérance aux pannes donnée par la fonction $loss$ appliquée à toute dépendance $dep(c_1, \dots, c_n)$ est donnée par l'expression:

$$loss(dep(c_1, \dots, c_n)) = \min(loss(c_1), \dots, loss(c_n))$$

3.2.2. La Stratégie Duplication. Cette stratégie sert à représenter deux composants (ou ensembles de composants) du système qui offrent le même service et qui peuvent être utilisés de façon indépendante.

Définition 6. Soit c_1 et c_2 deux composants (ou ensemble de composants) du système. Alors une duplication entre c_1 et c_2 est définie par $dup(c_1, c_2)$.

Hypothèse 2. Il existe un connecteur actif du type Choose auquel c_1 et c_2 sont liés.

Hypothèse 3. Il n'y a aucun connecteur ou composant actif (autre qu'un connecteur du type Choose) entre les composants qui constituent c_1 et ceux qui constituent c_2 .

Hypothèse 4. Aucun composant qui constitue c_1 n'est présent dans la constitution de c_2 . Aucun composant qui constitue c_2 est présent dans la constitution de c_1 .

Ces hypothèses servent à garantir que c_1 et c_2 fournissent le même service au même composant du système, introduisant une redondance, ainsi qu'à établir l'indépendance entre c_1 et c_2 . Deux composants sont dits indépendants si la panne d'un des composants n'affecte pas l'autre composant.

Propriété Garantie 2. La tolérance aux pannes donnée par la fonction $loss$ appliquée à toute duplication $dup(c_1, c_2)$ est donnée par l'expression:

$$loss(dup(c_1, c_2)) = loss(c_1) + loss(c_2)$$

3.3. Vue Sûreté de Fonctionnement. Pour vérifier l'exigence de tolérance aux pannes nous construisons une vue SdF qui lui correspond. Une vue SdF est créée par l'application successive de stratégies de tolérance aux pannes, et est représentée sous la forme d'un arbre d'application de stratégies dans lequel les nœuds sont donnés par les stratégies appliquées, et les feuilles par les composants du système. À ce niveau l'architecture du système doit déjà exister. Par contre, pour construire les vues SdF, nous n'avons besoin que de l'information sur les composants disponibles, et pas des liens entre eux.

Pour aider à la compréhension de la création des vues SdF, nous allons maintenant créer la vue qui correspondent à l'exigence SdF introduite dans la Section 3.1 sur les composants de l'architecture de la Figure 3.

3.3.1. Vue SdF pour l'Exigence 1. L'exigence 1 dit que le système hydraulique vert doit tolérer la panne d'un des deux moteurs. Cela veut dire que, même si un des deux moteurs tombe en panne, le système arrive quand même à fournir de la puissance hydraulique.

Pour la construction de cette vue SdF nous commençons par créer une duplication entre les fonctions $ENG1$ et $ENG2$, vu que c'est au niveau de ces fonctions que l'exigence est basée. Plusieurs décompositions des fonctions du système peuvent être effectuées. L'important est que la décomposition proposée satisfasse l'exigence. Pour cette exigence, nous proposons la décomposition suivante:

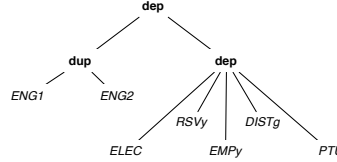


FIG. 4. Une vue SdF pour l'exigence 1

Les stratégies appliquées sont représentées en **gras**, et les fonctions du système utilisées dans la construction de la vue SdF sont représentées en *italique*.

Pour la vérification de l'exigence sur cette vue SdF il faut tenir compte de l'impact de la perte d'un des deux composants spécifiques. Dans ce cas nous considérons la duplication comme étant un seul élément du système, *c-à-d*, $loss(dup(ENG1,ENG2)) = 1$.

L'exigence est vérifiée si $loss(Vue_SdF_Exg1_SysHyd) = 1$, *c-à-d* si, dans le cas où nous ne disposons que d'un seul moteur, il faut quand même une panne en plus pour faire tomber la distribution. En utilisant les propriétés garanties par les différentes stratégies utilisées nous obtenons:

$$loss(Vue_SdF_Exg1_SysHyd) = \min(\begin{aligned} &loss(dup(ENG1,ENG2)), \\ &loss(dep(ELEC,RSVy,EMPy,DISTg,PTU)) \end{aligned})$$

En effectuant le calcul nous pouvons vérifier que le résultat est celui souhaité, et donc que la vue SdF proposée satisfait bien l'exigence 1.

3.4. Lien entre les Vues SdF et la Vue Architecturale par Intermédiaire des Zones. Dans cette section, nous présentons le lien entre la vue SdF et la vue architecturale du système en introduisant la notion de *zone*. C'est à travers des *zones* que les exigences de tolérance aux pannes utilisées pour créer les vues SdF vont être vérifiées.

Définition 7. Une zone est une entité abstraite composée par des composants et des connecteurs de l'architecture, et qui représente une partie du système. Les connecteurs entre deux composants d'une même zone appartiennent à la zone aussi. Les connecteurs entre des composants de deux zones différentes n'appartiennent pas à la zone.

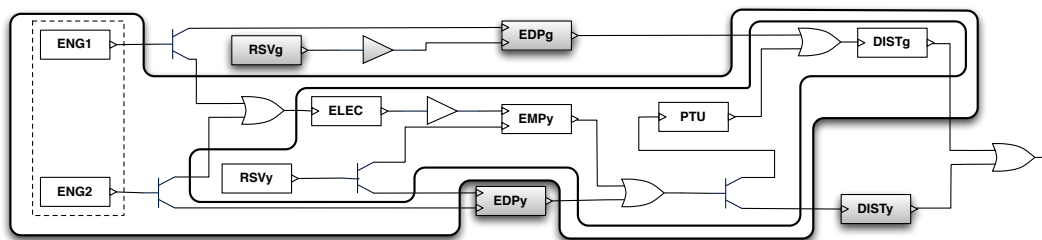


FIG. 5. *Vue Architecturale Correspondant à l'Exigence 1*

Pour créer la vue architecturale correspondant à une exigence nous utilisons la vue SdF de cette même exigence. Comme défini dans la Section 3.2, chaque stratégie est basée sur certaines hypothèses. C'est au niveau de chaque zone que ces hypothèses doivent être vérifiées.

3.4.1. *Vue Architecturale pour l'Exigence 1.* La génération d'une vue architecturale pour une exigence consiste à parcourir l'arbre qui représente sa vue SdF, et à introduire dans l'architecture une zone pour chaque stratégie utilisée. La vue architecturale obtenue pour l'exigence 1 est présentée dans la Figure 5. Dans cet exemple les hypothèse de la stratégie dépendance sont facilement vérifiées. Pour la zone originaire d'une duplication entre les composants *ENG1* et *ENG2* il faut vérifier les hypothèses d'application de la stratégie:

Preuve (Hypothèse 2). *Les composants ENG1 et ENG2 sont tous les deux connectés au même connecteur Choose (ce qui est visible dans l'architecture). L'hypothèse est donc vérifiée.*

Preuve (Hypothèse 3). *Dans l'architecture il n'existe aucun autre connecteur que Choose entre ENG1 et ENG2. L'hypothèse est ainsi vérifiée.*

Preuve (Hypothèse 4). *Les composants ENG1 et ENG2 sont deux composants différents. L'hypothèse est donc vérifiée.*

Vu que toutes les zones satisfont bien les propriétés de la stratégie utilisée pour les construire, il est possible de dire que la vue architecturale proposée est correcte et correspond à la vue sûreté de fonctionnement qui satisfait l'exigence 1. Ceci veut dire que la vue architecturale de base satisfait l'exigence 1.

4. COMPARAISON ENTRE APPROCHES

La sûreté de fonctionnement est classée en différents domaines. L'étude de la *fiabilité* d'un système, offre la possibilité de quantifier la qualité d'un système par rapport à sa probabilité de défaillance. La *tolérance aux pannes* introduit les méthodologies qui permettent l'augmentation de la fiabilité d'un système. L'objectif de ce travail se base dans la vérification de certaines propriétés de sûreté de fonctionnement sur une architecture d'un système. Ainsi, les travaux sur les techniques de tolérance aux pannes, notamment sur les modèles d'architecture pour la tolérance aux pannes [Bru09] sont essentiels pour notre travail car il nous permettent la définition de nouvelles stratégies de tolérance aux pannes.

Nous ne nous comparons pas aux travaux qui proposent des ADL et des méthodes pour introduire des exigences fonctionnelles dans le système, comme les ADL Darwin [MDEK95, MK96], Rapide [LV95] ou ACME [Garoo].

Le noyau de notre travail est dans la vérification d'exigences de sûreté de fonctionnement sur les architectures. Les approches principales, qui nous ont inspiré pour ce travail, sont les travaux sur la sûreté de fonctionnement en utilisant les langages AltaRica [APGRoo, Poioo] et AADL [FGHo6].

AltaRica permet la formalisation du comportement des systèmes en présence de pannes, ainsi que l'analyse de modèles en utilisant les nombreux outils disponibles. La vue dysfonctionnelle d'un modèle AltaRica est possible grâce à l'introduction d'événements qui modélisent la défaillance des composants. Sa capacité à réaliser des modèles compositionnels et hiérarchiques lui permet de modéliser des systèmes complexes. Notre but, par contre, c'était de proposer des vérifications de SdF, éventuellement avec des résultats moins précis que ceux obtenus avec AltaRica, sur des architectures simples. Les modèles AltaRica décrivent tout le système en détail, y compris son fonctionnement, tandis que nous imposons juste les différents composants du système et les liens entre eux.

Les travaux présentés par Kehren dans [Keh05] se basent sur la définition du comportement des systèmes en présence de pannes, modélisés en utilisant AltaRica. Il propose des méthodes qui assistent la modélisation et évaluation qualitative de l'architecture de sûreté de fonctionnement de systèmes embarqués complexes. La partie de son travail la plus proche du nôtre se base dans la validation d'exigences à base de motifs. Pour faire cette vérification il procède à la détection de motifs dans l'architecture et, après, à la vérification de la satisfaction des propriétés internes et d'environnement. Notre approche est très similaire. La grande différence se trouve dans l'identification des motifs dans l'architecture, que, dans notre cas, nous obtenons à partir des stratégies de tolérance aux pannes utilisées dans l'analyse de SdF. La vérification des exigences dans son approche est beaucoup plus formelle et se base sur le raffinement d'architectures, tandis que notre approche se base sur des propriétés très simples à vérifier sur les composants et connecteurs utilisés dans chaque stratégie.

Le langage AADL [FGHo6] permet la conception et l'analyse de systèmes complexes, critiques et temps réel dans plusieurs milieux industriels. Le langage comporte une extension qui sert à décrire les caractéristiques du système modélisé liées à la sûreté de fonctionnement [SAo6]. Pour ça ils définissent un sous-langage qui peut être utilisé pour déclarer des modèles d'erreur et les associer aux composants de la spécification de l'architecture.

Dans ses travaux [Rug07], Rugina propose le guidage de l'élaboration des modèles AADL de sûreté de fonctionnement par une méthode itérative, qui prend en compte progressivement les dépendances entre les composants.

Cette approche est itérative, *c-a-d* à chaque itération des nouvelles informations sont ajoutées. Dans un premier temps chaque composant du système est décrit, en introduisant leur comportement face aux pannes. Ensuite, des dépendances de plusieurs types sont définies entre les composants. Les vérifications de SdF sont faites au niveau de ces dépendances. Dans notre travail, nous utilisons une approche dans laquelle l'architecture est séparée de l'analyse SdF. Les vérifications sont effectuées d'une part au moment de l'analyse SdF, et d'une autre au moment de la vérification de l'architecture. Ceci implique une connaissance minimale de l'architecture au moment de l'analyse sûreté de fonctionnement, sans, pour autant, avoir besoin que l'architecture soit complètement définie.

Dans toutes ces solutions les propriétés de sûreté de fonctionnement sont vérifiées sur des modèles et/ou éléments d'architecture complètement définis. La définition du comportement des différents éléments du langage est souvent nécessaire. Ainsi, des analyses très précises sont effectuées au détriment de la complexité des calculs. La principale différence entre les approches présentées et la nôtre réside dans l'aspect complexité. Dans notre approche le but n'est pas de fournir une solution précise, mais de vérifier si une solution est possible ou pas, de façon à aider le processus d'analyse de sûreté de fonctionnement. Notre analyse peut être effectuée très tôt dans le développement, vu que seulement une connaissance de base de l'architecture est nécessaire.

5. CONCLUSIONS ET PERSPECTIVES

Avec ce travail, nous proposons une méthodologie pour vérifier des exigences de tolérance aux pannes, dans des architectures de systèmes. Pour ça, nous définissons deux vues pour chaque exigence:

une vue sûreté de fonctionnement et une vue architecturale. En faisant certaines vérifications au niveau de chaque vue, il est possible de déterminer si une architecture satisfait ou pas une certaine exigence.

Ce travail est encore en cours de développement, et les améliorations à apporter prochainement sont nombreuses. Un des premiers points à faire évoluer est l'inclusion de pannes au niveau des connecteurs. Pour le moment nous considérons que seuls les composants peuvent introduire des pannes dans le système. Cela nous a permis de définir une sémantique préliminaire de propagation des pannes pour chaque connecteur défini, et de tester ainsi notre méthode. Pour inclure l'introduction de pannes par les connecteurs il faudra changer leur sémantique pour que la propagation des pannes considère aussi des pannes internes.

Pour le moment nous ne proposons que deux stratégies de tolérance aux pannes pour construire la vue sûreté de fonctionnement. Cette bibliothèque de stratégies devrait être augmentée pour pouvoir répondre à plus d'exigences. Nous envisageons d'étudier les stratégies pertinentes pour tolérer la corruption des données telles que l'architecture COMMAND-MONITORING ou la triplification avec vote. Une vérification formelle des hypothèses associées à chaque stratégie et vérifiées par chaque zone est prévue.

Un autre ajout à ce travail sera le développement d'un outil qui permet la définition d'architectures en utilisant l'ADL proposé dans cet article, et qui permet de créer les vues SdF et architecturales, ainsi que la vérification des différentes propriétés sur les deux vues.

RÉFÉRENCES

- [APGR00] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The AltaRica Formalism for Describing Concurrent Systems. *Fundam. Inform.*, 40(2-3):109-124, 2000.
- [Bru09] Julien Brunel. New Platform Concept - Modélisation AltaRica d'un modèle PAM et Utilisation de Motifs d'Architecture. Technical Report RT-DTIM 2/14050, Onera - The French Aerospace Lab, Juillet 2009.
- [FGHo6] Peter H. Feiler, David P. Gluch, and John J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical report, Software Engineering Institute, 2006.
- [Gar00] Garlan, David and Monroe, Robert T. and Wile, David. Acme: Architectural Description of Component-Based Systems. In Gary T. Leavens and Murali Sitaraman, editor, *Foundations of Component-Based Systems*, pages 47-68. Cambridge University Press, New York, NY, USA, 2000.
- [Keh05] Christophe Kehren. *Motifs formels d'architecture de systèmes pour la sûreté de fonctionnement*. PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace, Décembre 2005.
- [LV95] David C. Luckham and James Vera. An Event-Based Architecture Definition Language. *IEEE Trans. Software Eng.*, 21(9):717-734, 1995.
- [MDEK95] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying Distributed Software Architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137-153, London, UK, 1995. Springer-Verlag.
- [MK96] Jeff Magee and Jeff Kramer. Dynamic Structure in Software Architectures. In *In Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1996.
- [Poi00] Gérald Point. *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, LaBRI, Université Bordeaux I, Janvier 2000.
- [Rug07] Ana-Elena Rugina. *Modélisation et évaluation de la sûreté de Fonctionnement - De AADL vers les réseaux de Petri Stochastiques*. PhD thesis, Institut National Polytechnique de Toulouse, Novembre 2007.
- [SA06] SAE-AS5506/1. *Annex E: Error Model Annex*. SAE Architecture Analysis and Design Language (AADL) Annex Volume 1, 2006.
- [Sago8] Laurent Sagaspe. *Allocation sûre dans les systèmes aéronautiques: Modélisation, Vérification et Génération*. PhD thesis, Université Bordeaux I, Décembre 2008.
- [Sto99] Neil Storey. Design for Safety. In *Towards System Safety: Proc. 7th Safety-Critical Systems Symposium*, pages 1-25, 1999.

ONERA/DTIM, 2, AVENUE ÉDOUARD BELIN, BP74025,, 31055 TOULOUSE CEDEX 4, FRANCE
E-mail address: `firstname.lastname@onera.fr`

ONERA/DTIM, 2, AVENUE ÉDOUARD BELIN, BP74025,, 31055 TOULOUSE CEDEX 4, FRANCE
E-mail address: `firstname.lastname@onera.fr`

ONERA/DTIM, 2, AVENUE ÉDOUARD BELIN, BP74025,, 31055 TOULOUSE CEDEX 4, FRANCE
E-mail address: `firstname.lastname@onera.fr`

ONERA/DTIM, 2, AVENUE ÉDOUARD BELIN, BP74025,, 31055 TOULOUSE CEDEX 4, FRANCE
E-mail address: `firstname.lastname@onera.fr`