# Evaluation d'une méthode de Génie Système pour l'Avionique Modulaire

Emmanuel Ledinot
Dassault Aviation
78 Quai Marcel Dassault
92552 Saint-Cloud Cedex

Gérard Le Lann
INRIA Rocquencourt
BP 105
78153 Le Chesnay Cedex

## 1. Introduction

Cet article résume les étapes et les résultats principaux d'une étude de faisabilité visant l'application de la méthode de Génie Système TRDF au développement des futurs systèmes d'avionique modulaire. Cette méthode est développée par le projet Reflecs de l'INRIA. L'étude a été soutenue par la DRET dans le cadre du contrat 94.34.395 en cotraitance entre Dassault Aviation et l'INRIA.
Nous commençons par rappeler brièvement les objectifs de l'avionique modulaire et le contexte de l'étude (sections 2 et 3). Nous résumons ensuite les trois étapes de l'étude de faisabilité : capture du besoin (section 4), conception d'une solution algorithmique et étude de ses propriétés (section 5), maquettage de la solution algorithmique et de l'outil de dimensionnement associé (section 6). Enfin nous donnons les conclusions de l'étude et ses prolongements envisagés.

## 2. Objectifs de l'avionique modulaire

L'avionique modulaire tend à faire disparaître la notion d'équipement propriétaire pour lui substituer celle de composant standard interopérable (tant matériel que logiciel). Elle vise le développement de systèmes pour lesquels le choix des ressources matérielles (modules de type carte) et le placement de l'applicatif soient beaucoup plus flexibles. L'élément nouveau par rapport aux systèmes d'avionique actuels est que le placement et le dimensionnement doivent devenir de véritables degrés de liberté de la conception. Les ressources matérielles sont modularisées pour pouvoir être exploitées de façon flexible et ainsi assurer une meilleure disponibilité des systèmes.

Concevoir des systèmes distribués temps-réel dur de façon générique, donc pour tous les placements possibles, est un problème ardu. En effet, un changement de configuration matérielle, de configuration logicielle, ou de placement du logiciel sur le matériel affecte directement l'ordonnancement des traitements sur chaque unité de calcul, les échanges sur les réseaux qui les relient, donc la vitesse de propagation des informations, etc. Or toute modification de l'ordonnancement local, ou des délais de communication entre les modules applicatifs, peut affecter profondément le comportement global du système.

L'avionique modulaire requiert donc de trouver un moyen de rendre le comportement *logique* de l'applicatif *invariant* par changement de ressource et de placement (valeurs des résultats de calcul et ordre d'apparition des observables) et de rendre le comportement *physique* de l'applicatif *prédictible* en fonction des performances de l'architecture matérielle support. Ceci est possible à condition de se donner un middleware (ou support exécutif d'application) conçu pour obtenir ces deux propriétés.

## 3. Contexte de l'étude

La méthode de Génie Système proposée par l'INRIA (appelée TRDF) a pour but de définir rigoureusement cette couche middleware répartie qui se situe entre le logiciel d'application et le système d'exploitation des différents sites de la configuration matérielle [1]. Cette couche a pour fonction de proposer à l'applicatif un modèle d'exécution indépendant du placement assurant des services de communication entre sites, d'ordonnancement temps-réel au niveau système pour tenir des échéances strictes sur des traitements répartis, et des services de tolérance aux fautes (détection de défaillance, reconfiguration, masquage).

## 4. Capture du besoin pour l'avionique modulaire

C'est la première étape de la méthode TRDF. Il s'agit de définir la spécification de la couche middleware à partir d'une analyse des besoins applicatifs. La sémantique du besoin n'est pas la même pour des systèmes de contrôle du trafic aérien ou pour des systèmes d'avionique modulaire. Mais dans les deux cas il s'agit de donner une spécification du besoin s'abstrayant totalement des spécificités fonctionnelles d'un applicatif particulier. Cette abstraction qui est

essentielle pour la factorisation des efforts de développement, repose sur des modèles et des propriétés couramment utilisés en informatique. La capture du besoin consiste à se donner des hypothèses (modèles) et des propriétés définissant l'applicatif.

Exemples d'hypothèses :

- modèle du code applicatif que le middleware va devoir exécuter (appelé modèle de taches). Dans cette étude il a été tenté de voir la totalité de l'applicatif comme un ensemble de transactions, une transaction étant définie par un graphe flots de données dirigé acyclique. Les noeuds représentent des appels de modules de code, les arcs représentent des communications. Quand plusieurs arcs arrivent sur un noeud, il y a attente (synchronisation) que toutes les informations soient arrivées pour faire l'appel. C'est un modèle événementiel. Les modules de code ont été supposés non interruptibles.
- modèle d'environnement qui consigne les hypothèses sur l'utilisation du système (lois d'arrivée des événements et densités maximales). Ces hypothèses conditionnent l'étude rigoureuse des temps de réponse pire cas. Dans cette étude il a été supposé que les événements déclenchant les transactions arrivaient selon des lois arbitraires avec des densités maximales spécifiées par un modèle à fenêtre glissante (pas plus de $n_k$ demandes d'activation de la transaction $T_k$ sur toute fenêtre de temps de longueur $w_k$).
- modèles des défaillances à tolérer. Pour l'avionique modulaire on s'est limité dans un premier temps à un modèle par arrêt pour les processeurs et à des défaillances par omission de messages pour les communications réseau. Les occurrences de défaillance suivent également une loi d'arrivée arbitraire et ont une densité maximale spécifiée selon un modèle à fenêtre glissante.

Exemples de propriétés :

- *sérialisabilité* des transactions. Cette propriété permet d'assurer que si chaque transaction vérifie des invariants quand elle s'exécute toute seule dans le système, ces invariants seront préservés même si elle s'exécute en parallèle avec d'autres transactions partageant des données avec elle.
- *ponctualité*. Chaque transaction est assortie d'une échéance stricte de terminaison au plus tard. Aucune échéance ne doit être violée.
- *détectabilité*. Toute défaillance de processeur doit être détectée en temps fini connu.

Par ailleurs existaient également des contraintes de mise en œuvre : complexité limitée des calculs des conditions de faisabilité (voir plus loin) et simplicité et efficacité a l'exécution. La taille des applicatifs à traiter était estimée à environ 1000 transactions.

# 5. Solutions algorithmiques INRIA et preuves

Ces solutions consistent en la spécification d'un modèle d'architecture fonctionnelle de type clients-serveurs et d'algorithmes distribués de contrôle de l'exécution d'un système asynchrone. Les preuves reposent sur l'étude théorique de propriétés de comportement qu'induisent ces algorithmes. Les preuves consistent à démontrer que l'on a bien la sérialisabilité, la ponctualité et la détectabilité, pour des conditions de faisabilité calculables. L'ensemble des algorithmes TR (temps-réel), TD (traitement distribué), TF (tolérance aux fautes) spécifiés par l'INRIA a été baptisé ORECA pour Ordonnancement Réparti par EChéance et Accord.

## *5.1 Algorithmes pour le traitement distribué*

La sérialisabilité [6] a pour but de faciliter la maîtrise du comportement global du système en contrôlant la modification de l'état d'un objet partagé par plusieurs transactions pour éviter les problèmes classiques d'écritures/lectures concurrentes sur de la mémoire partagée. Pour la solution retenue, le prix à payer pour la sérialisabilité est l'exécution d'un algorithme de consensus et le blocage de certaines transactions le temps que d'autres terminent et relâchent les objets partagés. Ces attentes peuvent aller à l'encontre du respect des échéances temps-réel. L'algorithme de consensus s'exécute en amont d'un ordonnancement distribué par lots non modifiables. Le consensus est fait de façon périodique. Il porte sur les demandes d'exécution de transactions non urgentes acquises pendant une période et postées par les clients vers les serveurs. Il a pour but de faire en sorte qu'à chaque «cycle système» tous les serveurs ordonnancent selon un ordre unique chacune des files d'attente (appelées lots) de demandes des modules des transactions activées. Comme ils appliquent la même stratégie d'ordonnancement et que les lots ordonnancés sont non modifiables, les objets partagés sur les différents serveurs du système sont partout modifiés comme si les transactions s'exécutaient séquentiellement, ce qui assure la sérialisabilité. L'algorithme de consensus est exécuté au début de chaque période.

### *5.2 Algorithmes pour le Temps Réel*

Deux classes de transactions ont été définies. Les transactions urgentes dont l'échéance relative est inférieure à 100 ms et les transactions non urgentes (les autres). Il est fait l'hypothèse que les transactions urgentes ne sont pas réparties sur plusieurs serveurs. Les serveurs appliquent la stratégie d'ordonnancement suivante :

- Earliest Deadline First (EDF) non oisif à réquisition de processeur pour les tâches urgentes au détriment des tâches non urgentes.
- EDF non oisif sans réquisition de processeur pour les tâches urgentes prises isolément
- algorithme oisif périodique à séquence de référence precalculée, sans réquisition de processeur pour les tâches non urgentes prises isolément (ordonnancement par lots non modifiables déclenché périodiquement après chaque consensus).

### *5.3 Algorithmes pour la Tolérance aux Fautes*

L'INRIA a étudié :

- l'effet des défaillances de ponctualité des horloges des serveurs sur le protocole de consensus quand celui-ci est réalisé de façon implicite (chaque serveur suit son horloge recalée sur un temps global construit à partir du temps local de chaque serveur, sans échanger d'information sur les demandes de transactions avec les autres serveurs), ainsi que le moyen de masquer ces défaillances.
- le moyen de tolérer les défaillances par omission dans le réseau.
- le moyen de détecter en temps fini borné les défaillances par arrêt des processeurs, le recouvrement étant assuré par la couche applicative.

### *5.4 Etablissement des conditions de faisabilité*

Les conditions de faisabilité de l'ordonnancement temps-réel établies par l'INRIA servent à définir une procédure (appelée « oracle ») qui, lorsqu'on lui soumet une quantification de l'applicatif et du système (durées pire cas d'exécution des modules des transactions placées, densité maximales d'arrivées des événements, des défaillances,...), répond oui ou non à la question : les échéances seront-elles toutes respectées ? Si la réponse est négative, l'oracle donne le ou les scénarios conduisant à la violation d'une échéance.

La principale difficulté a été posée par l'identification des scénarios pires cas vis-à-vis des arrivées événementielles et des différentes permutations de tâches pouvant être présentes dans les lots chez les serveurs. L'adversaire utilisé dans les raisonnements pour l'établissement des preuves, aux fins d'identifier les scénarios pires cas, déclenche le maximum (autorisé selon les densités maximales des arrivées arbitraires) de tâches urgentes et non urgentes aux moments les plus défavorables pour chacune des transactions prises tour à tour.
Les transactions étant distribuées sur les serveurs, il a fallu prendre en compte les synchronisations dues aux communications entre actions d'une même transaction placées sur des serveurs différents. Cette analyse utilise des calculs en algèbre (max,+) pour déterminer les chemins de calcul les plus longs.
D'autre part, bien que les densités des demandes d'activation de transactions soient bornées supérieurement, les tailles des files d'attente sont fluctuantes dans le temps, ce qui impose d'identifier des conditions de faisabilité pour des situations de surcharge temporaire d'un lot sur l'autre.
Pour plus d'information se reporter à [2].

## 6. Maquettage Dassault Aviation

Une plate-forme de simulation et d'instrumentation temps-réel n'était pas requise pour une étude de faisabilité comme celle-ci. Le maquettage a été divisé en deux parties :

1. une maquette comportementale sur réseau de stations de travail sous Unix reliées par un Ethernet non déterministe. Cette maquette permet une validation fonctionnelle hors temps-réel.
2. une maquette de dimensionnement comportant une réalisation de l'oracle et une émulation des comportements temporels pire cas grâce à un outil de modélisation de réseau à files d'attente, en l'occurrence Modline de Simulog. Le modèle Modline de l'exécution par Oreca d'un applicatif placé sur une architecture donnée a pour objectif de recouper, dans la mesure du possible, les résultats produits par l'oracle.

### *6.1 Configurateur*

Pour démontrer la généricité de la méthode de conception par rapport à l'architecture matérielle et au placement de l'applicatif sur cette architecture, un outil de génération de code, baptisé configurateur, a été développé. Il prend en entrée des fichiers de configuration décrivant les transactions du code applicatif, les durées d'exécution pire cas de leurs modules et leurs densités maximales d'arrivée, le nombre de serveurs et le placement de la configuration

considérée. A partir de ces fichiers, des modules de code applicatif et des modules de l'implantation d'Oreca, il génère un exécutable réparti sur le réseau de stations de travail. Il génère également des fichiers de données pour l'outil Modline afin de simuler des scénarios temporels pire cas pour la configuration considérée. Grâce à cet outillage différentes configurations matérielles ont pu être testées allant de 1 à 15 serveurs.

### *6.2 Maquette comportementale*

Elle a consisté à développer une fonction de conduite de tir air-air sous forme transactionnelle, à implanter Oreca au-dessus d'Unix et à simuler l'ensemble en observant la propagation des événements dans le réseau grâce à des visualisations graphiques. Le but était d'observer visuellement l'effet de l'ordonnancement et du contrôle de concurrence assurant la sérialisabilité. La maquette comportementale a permis de mettre en évidence une lacune dans la spécification de besoin du middleware et les effets de l'ordonnancement d'Oreca.

### *6.3 Maquette de dimensionnement*

L'oracle (cf. [2] et [5]) est un programme batch (un outil) qui vérifie que le système de contraintes que constituent les conditions de faisabilité est ou non satisfait. Cela revient à tester de façon optimisée un nombre de scénarios pire cas qui dépend des valeurs numériques des densités maximales et de la période de « scrutation système » (période de réalisation des consensus et d'ordonnancement). Le nombre de scénarios devant être testés peut aller de 1 à plusieurs centaines. Les performances de l'oracle ont été bonnes. Son maquettage a permis de mettre en évidence une sous-spécification des hypothèses de densité maximale de demandes de transactions conduisant à des avalanches pire cas irréalistes du point de vue fonctionnel.
Le modèle Modline a permis de recouper par la simulation l'analyse temporelle pire cas faite de façon algébrique par l'oracle (calcul matriciel (max,+)).

## 7. Conclusions et prolongements

Cette étude de faisabilité a donné des résultats intéressants et a permis d'identifier certaines extensions et améliorations nécessaires. Elle a permis à Dassault Aviation de se faire une première idée sur l'intérêt de la méthode TRDF proposée par le projet Reflecs de l'INRIA pour le développement des futurs systèmes d'avionique à architecture modulaire.
Elle a permis d'établir une première spécification d'un applicatif avionique modulaire générique et d'un exécutif distribué générique. Elle a produit une première solution algorithmique de type middleware répondant à cette spécification. Enfin, grâce au maquettage, elle a permis d'évaluer les apports de la méthode ainsi que les points positifs et moins positifs de la spécification du besoin, et de la solution algorithmique.

La méthode TRDF apporte un certain nombre d'éléments positifs :

- elle hisse au niveau système une démarche de conception rigoureuse qui n'existe aujourd'hui qu'au niveau équipement.
- elle propose une démarche hiérarchique de conception pour casser la complexité d'un grand système en plusieurs sous-problèmes de conception rigoureusement reliés entre eux (par des obligations de preuve analogues à celles rencontrées dans des méthodes formelles de développement de logiciel par raffinement comme la méthode B)
- elle apporte une formalisation et une classification des différents aspects comportementaux qui doivent être passés en revue dans la phase de capture du problème informatique générique (spécification de besoin du middleware)
- elle propose la notion de transaction sérialisée comme module de niveau système facilitant l'intégration et la validation d'un gros applicatif distribué. Cette approche ouvre une perspective intéressante en preuve de programme réparti.
- elle propose un modèle d'exécution distribuée qui permet de rendre le comportement de l'applicatif insensible (sur le plan fonctionnel) aux changements de placement. Ceci se fait au prix d'une analyse approfondie de l'applicatif (la définition des transactions) et de l'utilisation de mécanismes système qui, tout comme ceux d'un système d'exploitation, ont un coût d'exécution. Néanmoins les bénéfices obtenus (en validation voire certification) sont de nature à l'emporter nettement sur le coût supplémentaire à consentir.
- elle s'efforce de hisser au niveau système l'ordonnancement des traitements et la démonstration rigoureuse de la tenue des temps de réponse en établissant des conditions de faisabilité de l'ordonnancement dont on dérive un oracle de dimensionnement.

Si l'on en vient maintenant à la solution algorithmique Oreca et à l'oracle de dimensionnement résultant du premier exercice d'application de la méthode TRDF au cas de l'avionique modulaire, les résultats sont suffisamment positifs pour justifier une poursuite de

l'effort d'évaluation sur les points qui restent à clarifier, sans l'être suffisamment pour permettre une mise en application immédiate.
Les résultats positifs du maquettage sont les suivants :

- il a été possible de traiter l'application conduite de tir air-air sous forme transactionnelle, mais la définition des transactions a été assez difficile. Elle a nécessité un outillage non négligeable.
- l'implantation d'Oreca a été aisée.
- la vérification de l'invariance du comportement de l'applicatif lors de changement de placement a été probante.
- la démonstration de la généricité de la démarche de dimensionnement (tenue des échéances temps-réel) par rapport à l'architecture matérielle a été probante.
- la démonstration de la rapidité des calculs pour l'oracle a été probante.

Les résultats moins positifs du maquettage sont :

- le caractère peu performant de l'ordonnancement d'Oreca pour certaines conditions de placement. La sérialisabilité est obtenue à un coût en efficacité que l'on peut réduire de façon significative.
- le pessimisme qui peut être introduit par les majorations (invariants de tâches et de lots cf. [ 2]) utilisées pour l'établissement des conditions de faisabilité et l'optimisation des temps de calcul de l'oracle (cf. contrainte de traiter des applicatifs d'environ 1000 transactions). Ces majorations, dans certains cas de placement irrégulier, conduisent à des résultats de dimensionnement difficilement acceptables.

L'étape de maquettage a par ailleurs révélé la nécessité de deux compléments dans la définition par Dassault Aviation de la spécification de besoin du middleware via les modèles abstraits de l'INRIA :

1. le modèle de tâches doit être enrichi par la possibilité de spécifier que l'ordre d'activation de transactions appartenant à certains groupes doit respecter leur ordre relatif d'occurrence (c'est-à-dire leur ordre d'acquisition par un client).
2. le modèle d'environnement doit être enrichi par la possibilité de spécifier des contraintes d'espacement et d'exclusion mutuelle valables par groupes de transactions.

La prise en compte de ces deux compléments de spécification du problème générique constituant la spécification du support exécutif est nécessaire pour avoir une solution industriellement exploitable. Si le premier point n'est pas traité on peut observer des comportements incorrects de l'applicatif par rapport à ce qu'attend le pilote dès qu'il y a des procédures de dialogue comportant plusieurs commandes successives dans un ordre précis. L'ordonnancement temps réel doit respecter cet ordre.
Le point 2 vise à rendre plus réalistes les avalanches de traitement prises en compte pour garantir la tenue des temps de réponse en pire cas de charge.

Traiter ces deux compléments de spécification implique des modifications suffisamment profondes de l'algorithmique Oreca et des conditions de faisabilité pour qu'une étude complémentaire soit nécessaire. D'ores et déjà, Dassault Aviation et l'INRIA ont une assez bonne idée de ce que doit être la future variante d'Oreca. A cette occasion d'autres compléments vont être introduits, comme la coexistence sur un même serveur d'un applicatif transactionnel, et d'un applicatif non transactionnel, ces deux applicatifs pouvant partager des données. Comme la coexistence va être introduite, il n'y aura plus nécessité de fédérer tout l'applicatif en transactions. Le nombre de transactions va donc pouvoir diminuer grandement (quelques dizaines au lieu de quelques milliers) ce qui va permettre de relaxer la contrainte d'efficacité de l'oracle et donc de reconsidérer les techniques d'approximation et d'optimisation des temps de calcul. Une à deux années de travail sont donc encore nécessaires avant d'engager une nouvelle étape de prototypage sur un démonstrateur d'avionique modulaire grandeur réelle.

## 8. Bibliographie


[1] G. Le Lann, Proof-Based System Engineering for Computing Systems. ESA-INCOSE Conference on Systems Engineering, Noordwijk (NL), nov. 1997, 8 p.

[2] Algorithmique TR/TD/TF Oreca. Spécification des algorithmes et de l'oracle de faisabilité. Contrat DRET 94.34.395 Lot 3, deux rapports projet Reflecs, INRIA Rocquencourt, 8 janvier 1996.

[3] Conception Modulaire de Systèmes Temps Réel Distribués Tolérant les Fautes. Contrat DRET 94.34.395, Rapport de synthèse finale, Document Dassault Aviation DGT 68983 du 18.12.96

[4] Conception Modulaire de Systèmes Temps Réel Distribués Tolérant les Fautes. Contrat DRET 94.34.395, Rapport sur les

démonstrations Architectures Avioniques Avancées, Document Dassault Aviation DGT 69927 du 3.03.97

[5] Conception Modulaire de Systèmes Temps Réel Distribués Tolérant les Fautes. Contrat DRET 94.34.395, Rapport d'évaluation des études de dimensionnement, Document Dassault Aviation DGT 68695 du 22.12.96

[6] Bernstein et al., Concurrency Control and Recovery in Database Systems. Addison-Wesley Pub. ISBN 0-201-10715-5, 1987, 370 p.