Expérimentation du modèle Saturne sur CHORUS et COOL[1]

Campan Claire
Dassault Aviation
campan@dassault-avion.fr

Le Biannic Yann
Dassault Aviation
lebiannic@dassault-avion.fr

Singhoff Frank
CNAM/CEDRIC
singhoff@inf.enst.fr

Résumé

Les applications temps réels embarquées nécessitent aujourd'hui l'utilisation d'outils de modélisation performants et d'environnements d'exécution distribuée complètement maîtrisés. Saturne est un modèle d'exécution définit par le CERT pour les systèmes temps réel embarqués. Cet article étudie l'adéquation des services offerts par le système d'exploitation CHORUS et le bus à objets au standard CORBA de Chorus Système : COOL, par rapport aux besoins de Saturne.

Mots clefs

Système temps réel embarqué, Saturne, Esterel, CHORUS, COOL, Tolérance aux pannes, CORBA

1 Introduction

Aujourd'hui, la société DASSAULT-AVIATION spécifie les systèmes de ses avions et ce sont des équipementiers (Thomson, Sagem, Sextant Avionique, etc) qui réalisent ces équipements. Le choix des composants matériels et logiciels est propre à chaque équipementier. Cette hétérogénéité entraîne un surcoût de fabrication et de maintenance. L'avionique modulaire a pour objectif de réduire ce surcoût en définissant une architecture matérielle et logicielle permettant l'utilisation de logiciels dits "sur étagère". Par logiciels "sur étagère", on entend des composants logiciels standards pouvant être achetés dans l'industrie[2]. L'avionique modulaire doit aussi permettre de réduire le coût d'une mission en augmentant le nombre d'heures de vol sans maintenance. Pour cela, les systèmes des avions devront intégrer des mécanismes de tolérance aux pannes pour compenser d'éventuelles défaillances en vol.

Parallèlement à l'avionique modulaire, DASSAULT-AVIATION étudie des méthodes de modélisation de logiciels temps réel embarqués où les applications s'exécutent selon un modèle synchrone faible[6] élaboré par le CERT[3] : le modèle Saturne[9, 1, 23]. Les applications utilisent ici un bus à objets au standard CORBA[4][19, 30] pour communiquer. CHORUS[15] étant le système d'exploitation évalué dans le cadre des études d'avionique modulaire, il est également choisi pour ce travail dont l'objectif est de déterminer sous quelles conditions le modèle Saturne peut être mis en œuvre sur un système d'exploitation distribué temps-réel. Une étude des mécanismes de tolérance aux pannes ainsi que l'analyse de l'apport des technologies CORBA dans le

[1] COOL pour **C**HORUS **O**bject **O**riented **L**ayer.
[2] On évoque souvent ce concept par le terme de COTS (**C**ommercial **O**n **T**he **S**helf products).
[3] CERT pour **C**entre d'**E**tudes et de **R**echerches de **T**oulouse.
[4] CORBA pour **C**ommon **O**bject **R**equest **B**roker **A**rchitecture.

développement d'applications temps réel complète ces objectifs. Ce projet a débouché sur une implantation de Saturne sur CHORUS où les communications sont assurées par COOL[28, 27], le bus à objets au standard CORBA de Chorus système.

Nous commençons par présenter le modèle synchrone et le modèle Saturne dans la partie 2. La partie 3 est consacrée à la description du sous-système Saturne sur CHORUS/COOL ainsi que des prototypes réalisés. Puis, avant de conclure, nous décrivons dans la partie 4 les mécanismes de tolérance aux pannes envisagés sur Saturne.

2 Présentation de Saturne et des résultats déjà connus

Pour développer des applications temps réel, des langages spécifiques ont été créés : les langages synchrones. Ils sont basés sur la notion de systèmes réactifs qui fut introduite par D. Harel et A. Pnueli[22]. Par système réactif, on entend généralement un système qui réagit immédiatement à des entrées en provenance d'un environnement extérieur en fournissant des sorties instantanément. Les langages synchrones les plus connus sont Lustre[21], Signal[17] et Esterel[3, 4, 5]. Ces langages facilitent le développement d'applications temps réel déterministes et permettent de mettre en œuvre des preuves de propriétés logiques et temporelles[18, 8]. Le modèle synchrone fort repose sur un certain nombre de concepts :

- L'hypothèse du synchrone. On utilise un temps logique discrétisé. On parle alors d'instant d'activation. On considère qu'une exécution d'un noyau synchrone s'effectue dans *un seul instant*. Le programme reçoit à l'instant t ses entrées. Il réalise ses calculs et ses communications avant l'occurrence de l'instant $t + 1$. Dans ce sens, on peut considérer que le temps logique d'exécution d'un programme synchrone est nul.
- L'atomicité de l'exécution d'une transition du système réactif.

Le logiciel critique du système avionique doit posséder des propriétés de prédictibilité logique (ou correction) qui suppose une preuve formelle du comportement du système, et prédictibilité temporelle qui suppose une preuve du respect des échéances. Pour obtenir ces propriétés, le modèle proposé par Esterel consiste à séparer la partie contrôle de la partie dite "transformationnelle" d'un système. Le contrôle définit le comportement du système en fonction de l'occurrence des événements, c'est une partie typiquement réactive qui doit être logiquement prédictible. Le "transformationnelle" désigne les tâches de calcul, c'est la partie algorithmique du système. Les échéances des tâches de calcul doivent être prédictibles. Ce partage conduit à séparer la partie synchrone et la partie asynchrone d'un système, le synchrone contrôlant l'exécution de l'asynchrone. Un logiciel avionique peut ainsi être conçu comme un ensemble de noyaux réactifs contrôlant l'exécution de tâches asynchrones effectuant des calculs.

L'architecture matérielle du système avionique est multiprocesseur, se pose alors le problème de la distribution des noyaux réactifs. En effet, dans le modèle synchrone fort, les communications sont purement logiques. Dans une architecture distribuée, les temps de communication ne sont pas négligeables, le modèle synchrone fort ne peut donc pas être respecté. Cette distribution, également induite par des besoins de tolérance aux pannes, a conduit le CERT à élaborer Saturne (ou modèle "synchrone faible[1]"). Le modèle Saturne (voir figure 1) est constitué d'un ensemble de noyaux réactifs écrits en langage Esterel où chaque noyau réactif contrôle

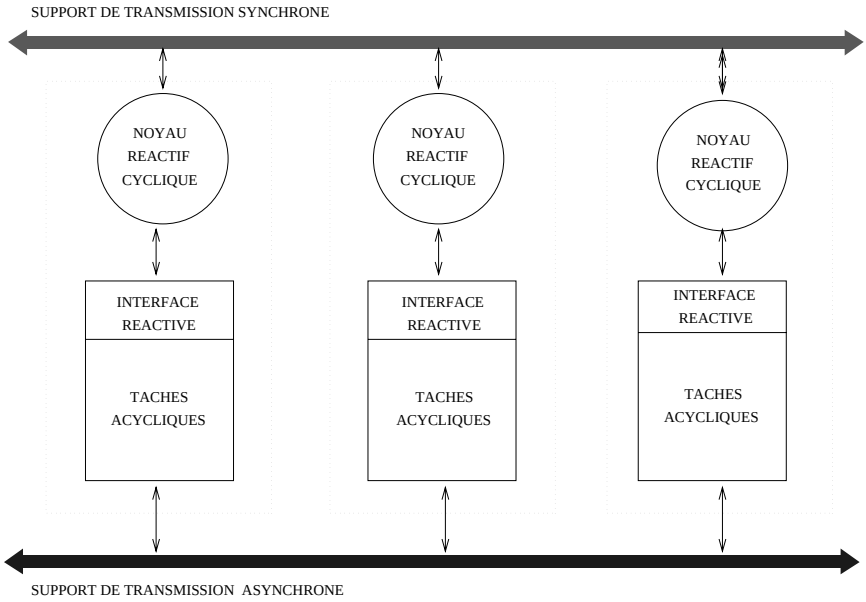

Figure 1: L'architecture de Saturne

un ensemble de tâches transformationnelles au travers d'une interface réactive. L'interface réactive offre au noyau synchrone les commandes lui permettant de consulter les résultats des tâches transformationnelles, de les créer, de les détruire ou de les suspendre. Le comportement d'un programme synchrone faible définit une suite d'instants équidistants durant lesquels le programme interagit avec son environnement (émission des sorties et réceptions des entrées). Au cours de l'intervalle séparant deux instants consécutifs ti et $ti + 1$, chaque noyau élabore sa réaction aux événements captés à l'instant ti. Cette suite d'instants définit une horloge qui cadence l'ensemble du système. Les signaux émis par un noyau sont captés par les autres à l'instant suivant. Le CERT et DASSAULT-AVIATION collaborent pour adapter ce premier modèle Saturne aux besoins spécifiques du logiciel avionique. En particulier, le modèle autorise aujourd'hui plusieurs fréquences d'activation des noyaux réactifs[24].

Le problème de la distribution de noyaux synchrones sur une architecture distribuée a fait l'objet de nombreux travaux. Citons A. Girault et C. Caspi[13] qui étudient des méthodes de répartition d'automates Esterel où les échanges entre processus sont synchrones et fiables. E. Nassor[18] propose une solution basée sur l'estampillage des signaux, il y inclut des mécanismes de tolérance aux pannes. Y. Faure[12] et F. Boulanger[7] intègrent les technologies objets au modèle synchrone. O. Potoniée[25] présente dans sa thèse un modèle d'exécution, basé sur le modèle Corea (modèle synchrone faible) où un algorithme de synchronisation d'horloge fournit une horloge globale qui active tous les objets réactifs de l'application dans un intervalle de temps connu et limité. G. Lelann[14] préconise un modèle transactionnel pour distribuer les noyaux synchrones, ce modèle permettant de prouver que les transactions respecteront leurs échéances.

3 Le sous-système Saturne sur CHORUS/COOL

3.1 Description du sous-système Saturne

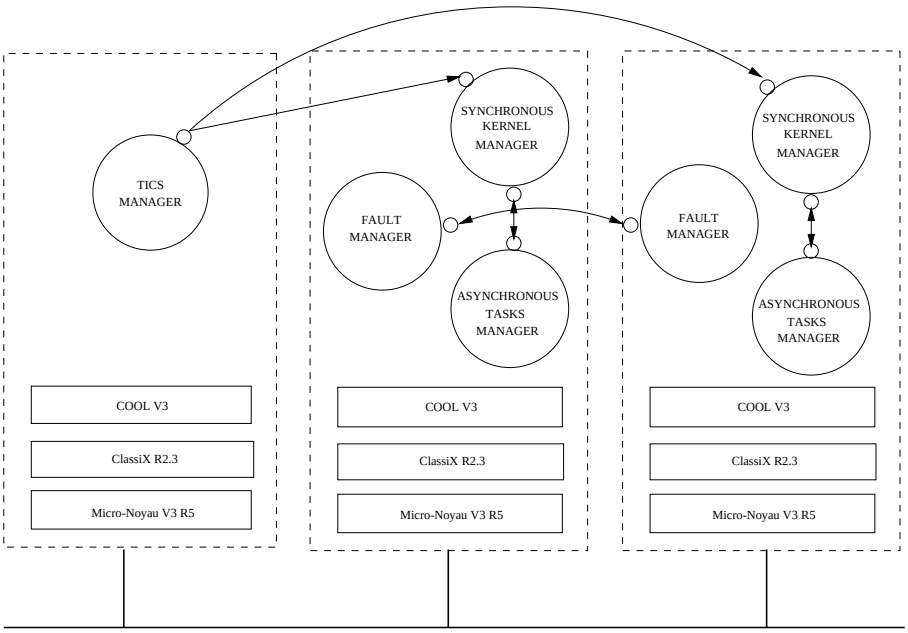

Figure 2: Le sous-système Saturne

Les abstractions de CHORUS et de COOL s'adaptent facilement à l'architecture de Saturne. Les composantes de Saturne sont réalisées sous forme d'acteurs CHORUS et de capsules COOL qui sont :

- Le TM (**T**ics **M**anager) : c'est l'acteur qui génère à intervalles réguliers des tops Saturne en direction des noyaux réactifs. Il implante l'horloge globale du modèle synchrone faible. Il y a un seul TM dans un système CHORUS. Le génération du top d'horloge *n'est qu'une simulation.* C'est pour cette raison que l'on se contente d'une diffusion CHORUS non fiable et non atomique. Dans un système réel, l'horloge de base devra être parfaitement fiable et précise et il est fort possible qu'une solution matérielle soit la plus adaptée à ce problème.

- L'AM (**A**synchronous tasks **M**anager) : cette capsule contient toutes les tâches transformationnelles, l'interface réactive de Saturne et un ordonnanceur pour les tâches transformationnelles. Chaque tâche transformationnelle est exécutée par une activité CHORUS. L'ordonnanceur est implanté d'une manière identique. L'interface réactive est un objet CORBA qui est invoqué d'une manière synchrone par le noyau réactif associé à l'AM. Toutes les tâches transformationnelles d'un même noyau réactif sont regroupées dans un unique AM. Enfin, d'un AM à un autre, l'ordonnanceur peut être différent.

- Le SM (**S**ynchronous kernel **M**anager) : cette capsule contient le code Esterel. Elle représente un noyau réactif du modèle Saturne. Elle reçoit régulièrement des tops en provenance du TM. A chaque top reçu, le SM active son automate Esterel, envoie ses commandes vers l'AM et diffuse ses signaux de sortie vers les autres noyaux synchrones.

- Le FM (**F**ault **M**anager) : cette capsule est responsable de la mise en œuvre des mécanismes de tolérance aux pannes sur Saturne. Il y a un FM par site CHORUS.

Une application COOL est constituée d'objets serveurs et d'objets clients. Nous modélisons donc le sous-système Saturne comme un ensemble d'objets CORBA. Dans Saturne, on trouve deux types d'objets :

```
#include "COMPLEX.idl"
interface callIR {
        void start-T1(out long identificateur, in long p1, in long p2);
        void start-T2(out long identificateur, in COMPLEX p1,
                in COMPLEX p2, in COMPLEX p3);
        void consult-T1(in long identificateur,out long p1);
        void consult-T2(in long identificateur,out COMPLEX p1);
        void kill(in long identificateur);
        void suspend(in long identificateur);
        void resume(in long identificateur);
        boolean isFinished(in long identificateur);
        boolean isSuspended(in long identificateur);
        boolean isActive(in long identificateur);
};
```

Figure 3: L'interface IDL d'une capsule AM

- Les objets de type "interface réactive". Dans chaque AM on trouve un objet de ce type. Il permet au noyau synchrone d'exécuter les commandes de contrôle sur les tâches transformationnelles de l'AM. Un exemple de code IDL[5] est donné dans la figure 3.

```
#include "HAUTEUR.idl"
interface NoyauInput {
        oneway void receiveMessage-APPUI-HOMING();
        oneway void receiveMessage-MODIFICATION-HS(in HAUTEUR data);
        oneway void receiveMessage-SELECTION-FORCAGEMER-ON();
};
```

Figure 4: L'interface IDL d'un noyau synchrone (capsule SM)

- Les objets de type "objets synchrones". Chaque objet de ce type encapsule un automate Esterel et exporte une interface IDL qui comprend tous les signaux d'entrée de l'automate. Un noyau *a* qui souhaite envoyer un signal *SIG* au noyau *b* invoquera la méthode *receiveMessage-SIG*() du noyau *b*. La figure 4 nous donne un exemple IDL d'un objet synchrone qui a trois signaux d'entrée.

[5] IDL pour **I**nterface **D**efinition **L**anguage.

3.2 Fonctionnement du sous-système Saturne

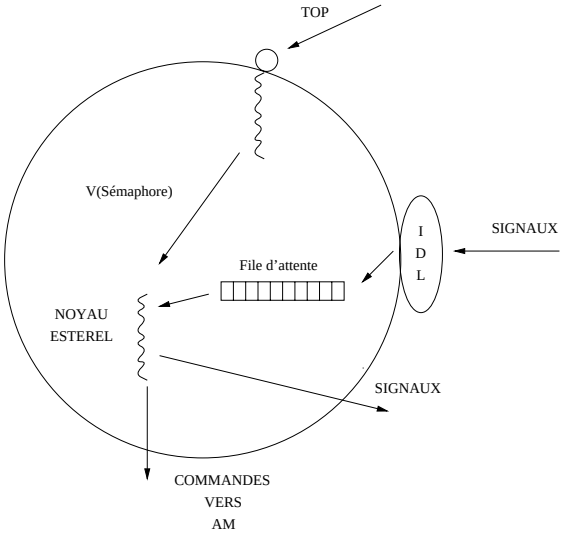

Figure 5: Description d'une capsule SM

Les capsules SM, comme le montre la figure 5, sont composées de :

1. Deux activités ; la première activité exécute le noyau Esterel. Elle est bloquée sur un sémaphore en attente des signaux d'entrée à consommer dans la file d'attente. La deuxième activité attend les tops émis par le TM. A chaque top, elle réveille le noyau Esterel.

2. D'une file d'attente qui permet de stocker les signaux d'entrée.

3. D'un objet CORBA qui exporte une interface IDL invoquée par les autres noyaux synchrones souhaitant envoyer un signal Esterel au noyau local.

Enfin, tout noyau synchrone possède une référence CORBA sur l'interface réactive de l'AM qui lui est associée et autant de références CORBA que de noyaux synchrones sur lesquelles il doit envoyer ses signaux de sortie.

La réception des signaux envoyés par les autres noyaux est continue. Chaque signal reçu est stocké dans la file d'attente et n'est pas immédiatement pris en compte par le noyau Esterel. L'activité en attente des tops est réveillée par le TM. A cet instant, elle regarde les signaux de la file d'attente qui peuvent être délivrés, les fournit au noyau Esterel, débloque celui-ci, puis, se met en attente du prochain top. L'activité du noyau Esterel, une fois débloquée, réalise une transition de l'automate Esterel pendant laquelle elle peut optionnellement invoquer des méthodes de l'interface réactive de son AM (d'une manière synchrone) et/ou émettre des signaux Esterel vers d'autres noyaux synchrones (de façon asynchrone). Puis on recommence un autre cycle au prochain top du TM.

La figure 6 montre l'exécution d'une application composée de n noyaux. T et $T+1$ représentent deux tops Saturne. Les noyaux synchrones sont tous activés en même temps lors de l'occurrence du top T. A cet instant, ils prennent en compte les signaux émis par les noyaux au top $T-1$, puis invoquent l'automate Esterel. Les automates Esterel émettent des signaux vers les autres noyaux et invoque les méthodes de leur interface réactive en fonction de leur besoin. L'exécution et les communications de *tous les automates Esterel* de l'application doivent

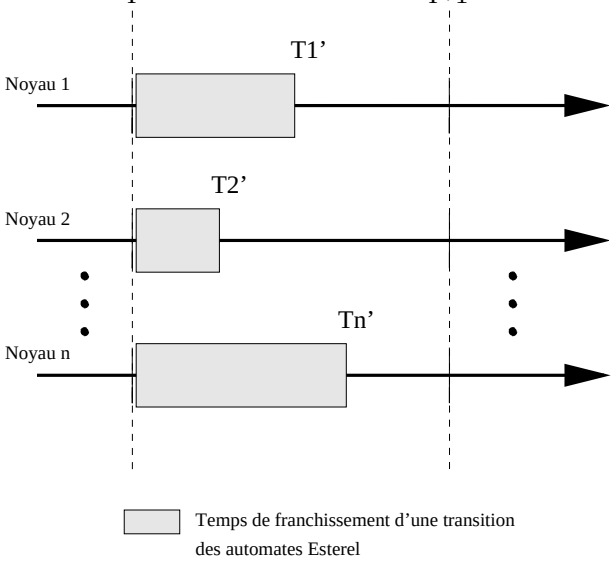

Figure 6: Fonctionnement du sous-système Saturne

être terminés avant $T+1$. Le temps entre T et Tn' représente la partie calcul des automates Esterel, le reliquat de temps de Tn' à $T+1$ est alloué à la communication des signaux Esterel et à l'exécution des tâches transformationnelles. Toute l'application Saturne est donc cadencée par le noyau le plus lent.

3.3 Prototypes réalisés et résultats obtenus

Afin de tester les mécanismes présentés dans les paragraphes précédents, nous avons développé deux prototypes. Le premier est constitué d'une application synchrone simple, fictive, comprenant trois noyaux Esterel et quatre tâches transformationnelles. L'objectif principal de ce premier prototype est la mise en œuvre des mécanismes de base de Saturne ainsi qu'une étude des performances et de l'adéquation des services offerts par CHORUS et COOL avec les besoins de Saturne. Une description détaillée de ce travail peut être consultée dans [10]. Les mesures de performances effectuées sur ce premier prototype portent sur ses réactions face à la répartition, sur la vérification que son comportement est conforme au modèle définit par le CERT ainsi que des mesures sur les invocations de méthodes COOL. Cette première implantation nous a permis de vérifier que CHORUS offrait tous les outils pour garantir une réactivité suffisante des noyaux synchrones ainsi que les outils permettant de gérer les tâches transformationnelles. Toutefois, les IPC CHORUS ne sont pas suffisamment déterministes. L'utilisation de COOL aggrave d'ailleurs cette situation. Trois causes semblent être responsables de l'indéterminisme des invocations de méthodes COOL :

- Les allocations dynamiques dans les souches et les squelettes CORBA.
- Les allocations dynamiques d'activités CHORUS pour traiter les invocations.
- Enfin, la principale cause est l'impossibilité pour le client, de maîtriser la priorité de l'activité qui va réaliser le service chez le serveur. Ce problème est illustré par la figure 7. Ici, la capsule A est celle de plus haute priorité, B celle de plus basse priorité et la priorité de C est intermédiaire avec celle de A et de B. Supposons qu'au départ, seul

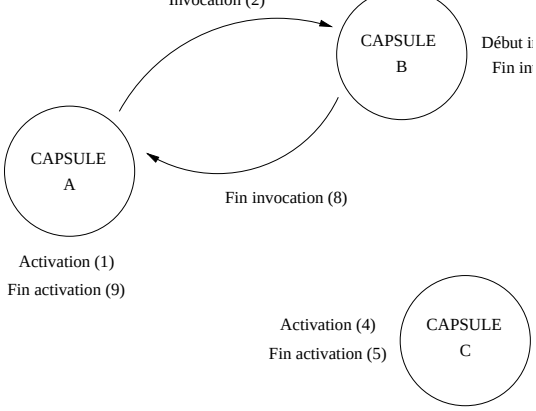

Figure 7: Déterminisme des invocations avec COOL

A et B existent. La capsule A, plus prioritaire, prend alors le processeur, puis réalise une invocation sur B. Durant cette invocation, A est bloquée en attente de la réponse et la capsule C est lancée. Puisque C est de plus forte priorité que B, on lui alloue le processeur. En effet, *avec COOL, A ne peut pas transmettre sa priorité à B.* De ce fait, C commence son exécution. L'invocation sur B ne reprendra que lorsque C sera terminée, alors que A était la capsule la plus prioritaire !

COOL n'offre pas suffisamment de maîtrise sur la manière dont sont réalisées les invocations de méthodes. En effet, le programmeur doit pouvoir maîtriser finement la gestion de la mémoire, la gestion des priorités et des activités réalisant les invocations (plutôt que de créer une activité pour chaque invocation, le programmeur peut vouloir décider la création, durant l'initialisation du serveur, d'un ensemble d'activités qui traitera toutes les invocations). La société Chorus système prépare actuellement une version de COOL permettant de solutionner ces deux problèmes. En dehors de ces limitations, COOL facilite la phase de développement et de test. L'utilisation des activités COOL est plus aisée que les activités CHORUS et il est évident que les services de synchronisation répartie et d'invocation sur groupes sont des outils intéressants. Enfin, nous verrons dans la partie suivante que les services d'invocation sur groupes de COOL peuvent être avantageusement utilisés.

Le deuxième prototype constitue une validation du premier : il utilise une application plus réaliste développée par DASSAULT-AVIATION. Celle-ci est constituée de sept noyaux synchrones communiquant à l'aide de 140 signaux. L'utilisation de cette deuxième application a démontré que le sous-système Saturne développé pour le premier prototype était exploitable quelque soit l'application synchrone utilisée.

4 Mécanismes de tolérance aux pannes sur Saturne

L'existence de fonctions critiques dans les applications qui devront fonctionner avec Saturne nécessite l'utilisation de mécanismes de tolérance aux pannes. Dans un système Saturne, on pourrait classer les fautes selon trois origines :

- Les fautes dues aux pannes d'une machine,

- Les fautes dues aux pannes d'un noyau synchrone,
- Les fautes dues aux pannes de tâches transformationnelles.

Il est difficile de déterminer les traitements à réaliser pour recouvrir une faute d'une tâche transformationnelle. En effet, les mécanismes à mettre en œuvre sont dépendants de la tâche elle-même. L'environnement d'exécution doit donc se contenter de fournir un cadre générique pour permettre à l'applicatif de définir ses mécanismes spécifiques. A ce titre, nous ne nous intéresserons pas ici aux tâches transformationnelles. Par contre, la panne d'un site ou d'un noyau synchrone n'empêche pas de définir des mécanismes standards de recouvrement qui fonctionnent quelque soit le code Esterel encapsulé. Les objectifs des mécanismes proposés sont donc de traiter les pannes franches de sites ou de noyaux synchrones ainsi que les pannes temporelles des noyaux synchrones. Un noyau synchrone provoque une panne temporelle lorsque ses traitements et ses communications déclenchés au top t ne sont pas terminés au top $t+1$.

Les mécanismes de tolérances aux pannes utilisés dans les systèmes temps réel utilisent presque toujours la redondance active. Delta 4[29] et MARS[26] en sont de bons exemples. Parfois, bien que cela soit rare, des mécanismes de points de reprise sont utilisés. C'est le cas de COSMOS[2] où le modèle flot de données est exploité afin d'obtenir des journaux de faible volume, et donc un temps de recouvrement accéléré. Dans un premier temps, nous nous proposons d'évaluer l'utilisation de la redondance active sur Saturne. Dans un deuxième temps, nous regardons comment il est envisageable d'implanter des mécanismes de point de reprise.

4.1 Utilisation de la redondance active

La redondance active permet de recouvrir *rapidement* des pannes de sites ou de noyaux synchrones. Ici, l'entité qui est répliquée est le noyau synchrone. Toutes les répliques d'un noyau synchrone sont rassemblées dans un groupe d'objets COOL. Le vote est effectué dans la capsule SM à chaque réception d'un top d'horloge, en comparant tous les signaux envoyés par les membres d'un même groupe de répliques. La méthode de vote qui détermine la valeur du signal considérée comme juste est accessible par l'utilisateur (l'utilisateur peut donc implanter un vote majoritaire ou tout autre politique comme celles décrites dans Electra[16]). L'utilisation de redondance active avec Saturne est beaucoup plus simple que dans les systèmes où les communications sont asynchrones. En effet, Saturne fournit une horloge globale, de ce fait, nous n'avons pas besoin d'algorithmes de diffusion possédant des propriétés d'ordre. Cependant, ceci ne nous dispense pas d'une diffusion atomique des signaux sur toutes les répliques du groupe. *Signalons que les noyaux Esterel sont très bien adaptés à la redondance active. En effet, le code Esterel a un comportement logique déterministe, ce qui est nécessaire dans ce cas[29].* Il n'y a bien sûr aucune garantie de déterminisme pour le code écrit en langage hôte par le programmeur.

4.2 Utilisation de points de reprise

Après l'utilisation de la redondance active pour augmenter le niveau de fiabilité d'une application Saturne, nous avons tenté d'y associer des points de reprise. Cette association permet de maintenir un nombre identique de répliques dans un groupe même en cas de défaillance. L'utilisation de points de reprise dans les systèmes temps réel est moins courante car elle ne permet pas un recouvrement des fautes aussi rapide que la redondance active. Toutefois, elle

peut être envisagée pour des fonctionnalités peu critiques du système, ou, comme dans COSMOS, quand des mécanismes accélérant le recouvrement de la panne peuvent être utilisés. Afin de diminuer le temps de recouvrement d'un noyau synchrone et de diminuer l'impact sur les performances de l'enregistrement des points de reprise, nous nous appuyons sur les deux observations suivantes :

- Faire exécuter une transition à un automate Esterel est une tâche qui est rapide et qui demande peu de ressource processeur.
- Les données véhiculées par les signaux Esterel semblent être de faible volume dans les applications d'avioniques[11].

Nous constituons nos points de reprise par l'enregistrement des signaux d'entrée délivrés à l'automate lors de chaque top. Lorsque l'on souhaite relancer un noyau synchrone, on lit le journal et on délivre les signaux d'entrée reçus en activant l'automate Esterel. Durant cette phase, les signaux en sortie ne sont pas réémis. Ces points de reprise sont donc particuliers puisqu'ils ne contiennent pas l'état de l'automate Esterel mais les informations qui permettent de le reconstituer rapidement. Il est possible, lors de la compilation, d'isoler les variables d'états de l'automate. On peut alors enregistrer régulièrement les valeurs de ces variables afin de constituer un véritable point de reprise. Ces deux techniques peuvent être avantageusement combinées. Dans tous les cas, l'enregistrement des points de reprise est facilité par Saturne grâce à son horloge globale qui synchronise tous les noyaux.

Avant de pouvoir relancer un noyau synchrone, il faut détecter sa panne. Nous utilisons une détection passive et active. La détection passive est réalisée par les noyaux synchrones. Les noyaux synchrones constituant l'application Saturne détecte les pannes lors du vote. Si lors d'un vote, ils constatent qu'ils leur manquent un signal (ils connaissent le contenu des groupes de redondance), ils savent qu'un noyau est victime, soit d'une panne franche, soit d'une panne temporelle. La détection passive peut entraîner une latence importante entre l'occurrence de la panne et la détection de la panne. En effet, un noyau ne reçoit pas systématiquement à chaque top un signal d'un autre noyau. De plus, avec l'utilisation de la détection passive seule, on ne détecte pas les pannes des noyaux n'émettant aucun signal. Aussi, nous utilisons un mécanisme de détection active qui consiste à rajouter des noyaux Esterel dont le seul but est de recevoir des signaux des noyaux de l'application Saturne. Ces noyaux de détection permettent de régler, si nécessaire, les deux problèmes ci-dessus.

Une fois la panne détectée, la capsule FM se charge de la recouvrir. Dans nos prototypes, il existe une capsule FM par site. Elles sont toutes rassemblées dans un groupe d'objets COOL (chaque FM exporte une interface IDL identique) On définit la notion de mode de fonctionnement. A chaque mode est associé une répartition différente des noyaux Esterel. En cas de panne d'un noyau seul, l'application reste dans le mode de fonctionnement courant. Lors de la panne d'un site, elle bascule dans un mode où il est prévu qu'aucun acteur ne s'exécute sur le site en panne. Lorsqu'un noyau détecte une panne, il invoque une méthode de l'interface de groupe des FM. Tous les FM effectuent ensemble, si nécessaire, le basculement dans le nouveau mode de fonctionnement. Chaque FM gère sur son site la création et la destruction de noyaux. La destruction d'un noyau intervient lors de la détection de sa panne si celui-ci est victime d'une panne temporelle. La création de noyau a lieu lors d'un basculement de mode, où tout simplement après détection de la panne d'un noyau seul. La détection de la panne d'un site

est réalisée par les FM qui considèrent un site en panne lorsque tous les noyaux du site de répondent plus. Les noyaux relancés s'exécutent hors du groupe tant qu'ils n'ont pas rattrapé l'état courant des autres répliques. Une fois complètement réinitialisés, ils réintègrent le groupe de vote. La panne est alors totalement recouverte.

4.3 Conclusions sur la tolérance aux pannes sur Saturne

Les deux paragraphes précédents ont donnés lieu à deux prototypes implantant, pour le premier, des mécanismes de redondance active, et pour le deuxième, des mécanismes de redondance active associés à des points de reprise. Le premier prototype a été entièrement réalisé. Il a permis de montrer que l'invocation sur groupe de COOL n'est pas à l'heure actuelle adaptée à la tolérance aux pannes. *Néanmoins, l'utilisation de COOL associée à celle de l'horloge globale de Saturne simplifie la conception et le développement d'algorithmes répartis.* De plus, COOL offre un mécanisme d'invocation sur groupe transparent : le client qui réalise l'invocation ne connaît pas la composition du groupe. Le code du client est donc indépendant du niveau de redondance. Le deuxième prototype n'a été que partiellement testé. Enfin, faute de temps, nous n'avons pas pu répondre précisément si ces mécanismes de tolérance aux pannes sont réellement compatibles avec des applications d'avioniques.

5 Conclusion et travaux futurs

Cette étude a montré la faisabilité de l'implantation sur CHORUS de noyaux réactifs contrôlant l'exécution de tâches transformationnelles et d'intégrant des mécanismes de tolérance aux pannes. Elle a permis d'apprécier la facilité et la rapidité de développement d'une application distribuée avec COOL (implantation de CORBA). Par contre la version de COOL utilisée, qui s'appuie sur le micro noyau CHORUS v3 r5 et ClassiX R2.3, ne permet pas de communication temps réel. Le modèle Saturne fait apparaître deux types différents de communication. Les communications entre noyaux ont besoin d'une communication rapide avec un respect des échéances. Les flux échangés sont faibles. Les communications entre tâches transformationnelles ont besoin d'un débit élevé mais les délais d'acheminement sont moins critiques. Une implantation correcte du modèle Saturne nécessite de disposer d'un système d'exploitation où cette dualité puisse être intégrée. Un travail similaire a déjà été réalisé dans le groupe ARCADE du CNET[6]. La suite de cette étude consistera a intégrer à la version R6 du micro noyau CHORUS un medium de communication adapté aux besoins de Saturne et à définir les propriétés d'un bus à objets adapté au logiciel avionique embarqué. On suivra les travaux portant sur les bus à objets temps réel, en particulier ceux du projet ReTINA et les éventuelles évolutions de COOL qui en découleront.

Nous tenons à remercier Eric Gressier pour ses nombreux conseils sur la réalisation de ce travail, Christine Ledey pour son aide sur le générateur de code et la mise en œuvre de l'interface homme-machine du deuxième prototype, Eric Nassor pour son soutien et ses nombreuses explications. Nous remercions aussi toutes les personnes du département des études logicielles de DASSAULT-AVIATION qui ont participés de près ou de loin à l'aboutissement de ce travail.

[6] Le support de communication choisi par le CNET est FDDI[20].

References

[1] F. Boniol M. Adelantado. Programming distributed reactive systems : a strong and weak synchronous coupling. CERT ONERA, September 1993.

[2] D. Cummings L. Alkalaj. Checkpoint/rollback in a distributed system using coarse-grained dataflow. Jet Propulsion Laboratory, California Institute of Technology, Proc. IEEE, 1994.

[3] G. Berry. The constructive semantics of pure esterel. Ecole des Mines de Paris, Sophia-Antipolis, 1995.

[4] G. Berry, P. Couronné, and G. Gonthier. Programmation synchrone des systèmes réactifs: le langage Esterel. *Techniques et Sciences de l'Informatique*, 6(4), 1987.

[5] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science Of Computer Programming*, 19(2):87–152, 1992.

[6] M. Adelantado F. Boniol. Programming communicating distributed reactive automata : the weak synchronous paradigm. CERT ONERA, International Conference on Decentralized and Distributed Systems, Palme de Mallorca, Spain, September 1993.

[7] F. Boulanger. Intégration de modules synchrones dans la programmation par objets. Thèse de doctorat, PARIS XI Orsay, décembre 1993.

[8] E. Audureau P. Enjalbert L. Farinas Del Cerro. Logique temporelle : sémantique et validation de programmes parallèles. Edition Masson, 1990.

[9] M. Adelantado F. Boniol M. cubéro-castan B. Lécussan R. Porche V. David. Projet SATURNE : modèle de programmation et modèle d'exécution pour un système temps réel d'aide à la décision. CERT ONERA, 5th Euromicro Workshop on Real-Time, Oulu, Finland, juin 1993.

[10] Singhoff F. Mise en oeuvre du modèle Saturne sur CHORUS/COOL. Mémoire d'ingénieur CNAM, Centre de Paris, septembre 1996.

[11] Singhoff F. Réalisation d'un sous-système Saturne tolérant les pannes sur CHORUS/COOL. Rapport de DEA Systèmes informatiques, Université PARIS VI, septembre 1996.

[12] Y. Faure. SATURNE objet : une approche distribuée orientée objet des systèmes temps réel mixtes réactifs/interruptibles. ENSAE, Rapport de DEA, juin 1996.

[13] P. Caspi A. Girault. Execution of distributed reactive systems. pages 15–26. in First international EURO-PAR Conference, LNCS 996, Springer Verlag, Stockholm Sweden, août 1995.

[14] G. Lelann. Algorithmique TR/TD/TF ORECA. Spécifications des algorithmes de l'oracle. Lot 3. Contrat DRET 94-395, 1994.

[15] F. Armand et al. M. Rozier, V. Abrassimov. Overview of the CHORUS distributed operating systems. Chorus Systèmes, February 1991. CS/TR-90-25.1.

[16] Silvano Maffeis. Run time support for object oriented distributed programming. Université de Zurich, February 1995.

[17] P. Le Guernic T. Gautier M. Le Borgne C. Le Maire. Programming real time applications with SIGNAL. INRIA-RENNES, Rapport numéro 1446, 1991.

[18] E. Nassor. Thèse de doctorat : Modélisation et validation d'applications temps réel distribuées. Université PARIS Sud, Centre d'Orsay, juin 1992.

[19] OMG TC Document 93.7.2. Object request broker architecture. 1993.

[20] A. Shah G. Pamakrishnan. FDDI, réseaux haut débit. Collection systèmes distribués, MASSON, 1994.

[21] N. Halbwachs P. Caspi P. Raymond D. Pilaud. Programmation et vérification des systèmes réactifs : le langage LUSTRE. Techniques et sciences informatiques, 1991.

[22] D. Harel A. Pnueli. On the developement of reactive systems. In Logic and Models of Concurrent Systems. Proc NATO Advanced Study Institute on Logics and Models for Verifications and Specification of Concurrent Systems, 1985.

[23] M. Adelantado F. Boniol R. Porche. Un modèle synchrone distribué et déterministe pour le temps-réel. CERT ONERA, mai 1993.

[24] M. Adelantado F. Boniol R. Porche. Etude d'un modèle hiérarchique et structuré multi-synchrone pour la programmation de systèmes réactifs. CERT ONERA, janvier 1996.

[25] O. Potonniée. thèse de doctorat : Etude et prototypage en ESTEREL de la gestion de processus d'un micro-noyau de système d'exploitation réparti avec garantie de service. PARIS VI/CNET, avril 1996.

[26] H. Kopetz H. Kantz G. Grunsteidl P. Puschner J. Reisinger. Tolerating transient faults in MARS. Proc. IEEE, January 1990.

[27] Chorus Systèmes. CHORUS/COOL-ORB Programmer's Guide. February 1996. CS/TR-96-2.1.

[28] Chorus Systèmes. CHORUS/COOL-ORB Programmer's Reference Manual. February 1996. CS/TR-96-3.1.

[29] D. Powell M. Chérèque P. Reymier J.L. Richier J. Voiron. Active replication in Delta-4. Proc. IEEE, 1992.

[30] T. J. Mowbray R. Zahavi. The essential CORBA. OMG Document, 1995.