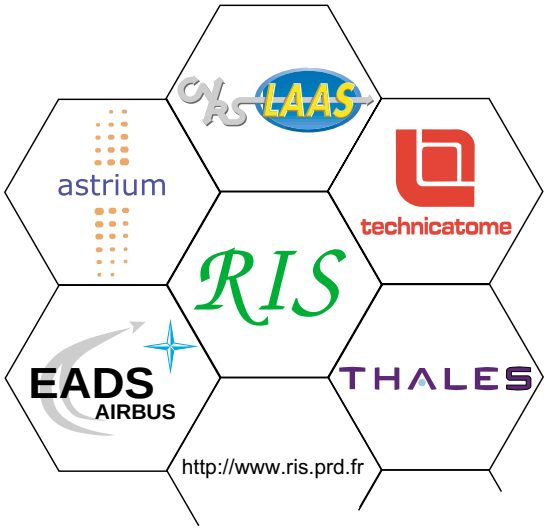

Réseau d'Ingénierie de la Sûreté de fonctionnement

Réunion n°3 du GT Logiciel Libre et Sûreté de Fonctionnement
20 septembre 2001 – LAAS-CNRS

Programme	
10 h00-12 h15	Introduction (approbation du compte-rendu précédent)
	Caractérisation de robustesse et de temps de réponse d'OS temps-réel, *Isabelle Puaut, IRISA*
	Analyse de modes de défaillance d'intergiciels CORBA, *Eric Marsden, LAAS-CNRS*
12 h 15-13 h 15	*Déjeuner*
13 h15-15 h15	Présentation de la bibliothèque de composants réutilisables destinés au système industriel Technicatome COMIT, *Philippe Coupoux, Technicatome*
	Logiciels et architectures libres à l'Agence Spatiale Européenne, *Eric Conquet, Jean-Loup Terraillon, ESA*
	Les logiciels libres dans les systèmes sol, *Patrick Pleczon, François Lecouat, Astrium*
15 h 15-15 h 30	*Pause*
15 h30-16 h30	Synthèse, mise à jour du planning des prochaines réunions et identification des contributions

Participants :

Membres du GT :

E. Conquet (ESA), P. Coupoux (Technicatome), Y. Crouzet (LAAS-CNRS), P. David (Astrium), Y. Garnier (SNCF), S. Goiffon (EADS Airbus), G. Mariano (INRETS), V. Nicomette (LAAS-CNRS), Y. Paindaveine (Commission Européenne), I. Puaut (IRISA), J-M. Tanneau (THALES), H. Waeselynck (LAAS-CNRS).

Absents ou excusés :

B. Bérard (LSV), J-L. Terraillon (ESA).

Invités :

J-C. Fabre (LAAS-CNRS), E. Marsden (LAAS-CNRS), P. Pleczon (Astrium).

1. Introduction

Le compte-rendu de la réunion du 11 juillet 2001 est approuvé avec quelques modifications. Il pourra être mis dans la partie publique du site web du *RIS*.

La position de THALES sur la brevetabilité du logiciel, brièvement présentée lors de cette dernière réunion, pourra être approfondie lors de la réunion qui traitera des aspects juridiques (Réunion 6, d'après le planning prévisionnel).

2. Architecture et validation

Caractérisation de robustesse et de temps de réponse d'OS temps-réel, *Isabelle Puaut, IRISA*

Les travaux présentés portent sur la caractérisation de supports exécutifs (basés sur des COTS ou LL) pour les applications temps-réel strict. L'exposé a comporté deux parties : 1) caractérisation expérimentale du comportement en présence de fautes, où l'on se concentre sur la validation de l'hypothèse de silence sur défaillance ; 2) évaluation analytique de pire cas de temps d'exécution (WCET).

La validation de l'hypothèse de silence sur défaillance est basée sur des expériences d'injection de fautes. L'injection procède par altération de mots mémoire (bit-flip simple), ce qui permet de simuler des fautes physiques. Les résultats présentés portent sur un noyau temps-réel COTS, Chorus. L'application de charge utilisée est une application fictive (suivi d'objets), mais des expériences ultérieures menées avec une application réelle, fournie par Dassault Aviation, ont donné des résultats similaires. Cette application fictive est une application répartie déterministe, ce qui rend possible l'implémentation d'un oracle global. Les expériences se sont déroulées sur une durée approximative de 3 mois. Pour chaque expérience, l'hypothèse de silence sur défaillance est considérée comme satisfaite si on observe soit un résultat correct de toutes les machines, soit un arrêt de la machine cible de l'injection (toutes les autres machines délivrant un résultat correct) : c'est le cas pour 80% des expériences, ce qui montre la nécessité de mécanismes additionnels pour améliorer la robustesse. Des expériences supplémentaires ont alors été conduites avec une couche middleware au-dessus de Chorus (middleware Hades). Hades, conçu et développé à l'IRISA, offre des mécanismes logiciels de détection et de confinement d'erreurs. Il comprend des mécanismes à base de redondance (ex : checksum, structures de données redondées), des contrôles temporels, des vérifications d'exécutions (ex : vérification en ligne du graphe d'appel, exécution de chaque service dans un espace mémoire séparé), et divers autres mécanismes basés sur la structure du code ou sa sémantique, et sur des contrôles de bas niveau (ex : erreur CPU de division par zéro). Grâce à cette couche middleware, la satisfaction de l'hypothèse de silence sur défaillance est observée pour 99% des expériences. Les 1% restantes montrent qu'une solution logicielle telle que Hades ne serait pas adéquate pour un système à forte criticité. En termes de coût, on double la taille mémoire nécessaire au support exécutif, et le temps d'exécution des appels système est plus que doublé. L'implémentation des mécanismes nécessite une instrumentation manuelle du code source de l'OS : on ne dispose donc pas d'une méthode automatique pour rendre un OS robuste. Selon les résultats expérimentaux, les mécanismes les plus efficaces sont la séparation mémoire des services, les contrôles bas niveau (erreur CPU), les contrôles de gestion des structures de données et les checksums.

La méthode employée pour l'évaluation de WCET est une méthode analytique basée sur l'analyse statique du code source. Elle présente l'avantage d'être sûre, mais pessimiste. Son application impose des restrictions sur le langage source : pas de récursivité, boucles bornées, pas d'appels dynamiques. En particulier, les langages orientés-objet posent problème, ce qui constitue une limitation pour l'analyse de certains COTS ou LL. De plus, on ne traite que des morceaux de code séquentiels. La méthode a été appliquée pour l'analyse de fonctions du noyau temps-réel RTEMS, moyennant quelques modifications du source C pour le rendre analysable. L'évaluation prend en compte des informations sur l'architecture matérielle (cache, pipeline), ce qui permet d'obtenir des résultats pas trop pessimistes. A titre d'exemple, les WCET estimés par l'analyse de RTEMS sont en moyenne 1,8 fois supérieurs aux WCET mesurés lors de tests activant des pires cas connus, ce qui reste raisonnable. Sans modélisation de la micro-architecture, les résultats sont significativement plus pessimistes, puisque le ratio estimé/mesuré est de 14.

Analyse de modes de défaillance d'intergiciels CORBA, *Eric Marsden, LAAS-CNRS*

Bien qu'il existe plusieurs travaux portant sur la caractérisation de supports exécutifs, les modes de défaillance des couches middleware restent peu étudiés comparés à ceux des noyaux ou OS. L'exposé porte sur l'analyse comparative de différentes implémentations CORBA, et propose une méthode expérimentale basée sur l'injection de fautes. L'objectif est double. Il s'agit d'une part d'obtenir des mesures permettant de guider le choix d'une implémentation parmi les alternatives existantes. D'autre part, l'analyse fine des modes de défaillance peut aider les intégrateurs à concevoir des wrappers pour améliorer la robustesse d'une implémentation cible.

Dans un système distribué basé sur CORBA, l'invocation de méthodes d'un objet distant conduit à traverser plusieurs couches logicielles et matérielles : se pose alors la question du niveau auquel injecter les fautes. De plus, le modèle de fautes sera différent selon que l'on s'intéresse aux fautes physiques, aux fautes logicielles, ou à des fautes

environnementales (ressources, niveau de charge). Enfin, la mise en œuvre du test de la robustesse d'un ORB est problématique, car la plupart des fonctionnalités offertes n'ont pas d'interface explicite.

Les travaux présentés ont ciblé le service de noms CORBA. Ce service possède une interface standard explicite, ce qui facilite la comparaison de différentes implémentations. L'application de charge activant ce service est une application fictive qui construit un graphe de noms, et cherche à résoudre des noms à partir de ce graphe. La construction du graphe est déterministe, ce qui facilite la mise en œuvre d'un oracle vérifiant les résultats du service de noms. Dans un premier temps, le modèle de fautes utilisé est la corruption de requêtes IIOP entrantes (IIOP = Internet Inter-ORB Protocol), par bit-flip simple ou double-zéro (2 octets consécutifs mis à zéro). On cherche ainsi à simuler des fautes transitoires du système de communication. La justification de ce modèle de fautes s'appuie sur des publications récentes mettant en évidence un taux d'erreurs non détectées par les checksums TCP, taux certes petit mais non négligeable compte tenu de la capacité des réseaux actuels (1 erreur toutes les 80 heures sur un LAN 100 Mb/s saturé). A noter que la corruption de requêtes IIOP peut aussi permettre de simuler des propagations d'erreurs depuis les courtiers distants, ou encore des interactions avec des clients malicieux.

Les expériences ont porté sur 4 implémentations CORBA : javaORB (COTS, Sun), omniORB (logiciel libre sous licence Gnu GPL, AT&T Lab.), ORBacus (COTS, Object Oriented Concepts), ORBit (logiciel libre sous licence Gnu GPL). Aucun cas de propagation d'erreur au niveau applicatif n'a été observé, ce qui est positif. Par contre, on observe une proportion non négligeable de "crash " du service de noms (le processus correspondant a été tué) et de "hang" du service (le processus existe toujours, mais ne répond pas à la requête dans un délai donné). Les résultats varient de façon significative selon l'implémentation testée : ceci n'est pas étonnant car la norme CORBA laisse un grand degré de liberté pour l'implémentation. La plus robuste est ORBacus, suivie de omniORB, qui totalisent le plus faible nombre de crash, hang et exception UNKNOWN. Une analyse fine des résultats en termes de position de la corruption bit-flip montre une grande sensibilité à certains bits de l'en-tête IIOP, et cette faiblesse est commune à toutes les implémentations. Ajouter un checksum au protocole Inter-ORB permettrait d'améliorer la robustesse vis-à-vis de fautes non malicieuses.

La méthode mise en œuvre est non intrusive, et facilement portable pour évaluer d'autres implémentation CORBA. Par contre, le modèle de fautes est limité : il faudra étendre l'analyse à d'autres types de faute. Une autre extension envisagée est d'étudier l'impact du système d'exploitation sous-jacent sur la robustesse.

Présentation de la bibliothèque de composants réutilisables destinés au système industriel Technicatome COMIT, *Philippe Coupoux, Technicatome*

Technicatome s'oriente vers des architectures à base de COTS pour le matériel (ex : Pentium) et l'OS (ex : QNX). Les systèmes visés sont des systèmes synchrones, à déroulement cyclique, et devant fonctionner 24H/24. Ces systèmes ont une longue durée de vie, d'où la nécessité d'une stratégie de pérennisation. Dans ce contexte, le projet COMIT vise à maîtriser les problèmes d'obsolescence des COTS, ainsi qu'à simplifier et harmoniser leur utilisation.

Le principe retenu est d'encapsuler les COTS, de façon à offrir aux applications la même interface quel que soit le composant effectivement utilisé. De la sorte, les modifications de code suite à un changement de COTS restent confinées, et les applicatifs ne sont pas affectés. Les fonctionnalités offertes aux applications ont été choisies une fois pour toutes, en se basant sur un modèle du matériel et du logiciel support nécessaires au fonctionnement d'un système temps-réel. Cela signifie qu'on n'utilise qu'un sous-ensemble des fonctionnalités du COTS. Par exemple, COMIT n'autorise qu'un seul schéma de communication inter-processus, l'échange de messages (sécurisés ou non) client/serveur, et ce même schéma est étendu pour le multiprocesseurs. Etant donnée une fonctionnalité à offrir, les éléments d'interface avec le COTS se basent sur les fonctions du composant retenues comme les plus adaptées, les plus robustes. La sur-couche qui encapsule le COTS offre également des mécanismes de tolérance aux fautes : des mécanismes spécifiques aux faiblesses du COTS, ainsi que des mécanismes génériques tels que chien de garde, code CRC des messages et des trames réseaux, contrôles d'échéances.

Concrètement, COMIT offre un cadre de développement orienté-objet, reposant sur une bibliothèque de composants réutilisables, classes et processus spécialisés. La bibliothèque de classes permet aux applications d'hériter de propriétés des applications de base COMIT : contrôles d'échéances automatiques, contrôles réguliers de l'intégrité du code, traitement centralisé des erreurs, schéma client/serveur orienté-objet. Des processus spécialisés gèrent les communications et les chiens de garde.

Ce cadre de développement est destiné à des applications de niveau B de criticité. L'architecture à base de COTS est une architecture triplex inspirée des résultats du projet ESPRIT GUARDS[1] (Generic Upgradable Architectures for Real-Time Dependable Systems). En ce qui concerne la sur-couche COMIT et les applicatifs, il n'y a pas de diversification logicielle car ce ne serait pas rentable vu le faible niveau de fiabilité des COTS.

[1] Voir par exemple http://www.wkap.nl/prod/b/0-7923-7295-6 ou http://computer.org/tpds/td1999/l0580abs.htm

Logiciels et architectures libres à l'Agence Spatiale Européenne, *Eric Conquet, Jean-Loup Terraillon, ESA*

Cet exposé a comporté deux parties :1) architectures libres à l'ESA et 2) expérience de validation de la sûreté d'un logiciel à partir du seul source.

L'ESA finance le développement d'applications présentant des caractéristiques récurrentes, d'où l'idée de mettre en place des architectures libres, ré-utilisables par les contractants. A noter que la notion d'architecture libre peut entrer en conflit avec les règles de l'Agence, qui doit promouvoir une diffusion européenne et non mondiale. A titre expérimental, deux projets ont été lancés pour les applications spatiales embarquées. Le projet OBOSS (http://spd-web.terma.com/Projects/OBOSS/Home_Page/) s'est intéressé aux systèmes de gestion des données de communication avec le sol (DHS, ou Data Handling System), et une implémentation conforme à la norme ESA PSS-07 a été proposée. Un autre projet mené à l'Université de Constance a permis de définir un "framework" orienté-objet pour la réutilisation d'architectures, dans le cas de systèmes AOCS (Attitude and Orbit Control System) pour le pilotage de satellites (http://www.softwareresearch.net/projects/AocsFrameworkProject/ProjectHomePage.html). Dans les deux cas, un problème majeur pour le passage à une utilisation opérationnelle en libre est le manque d'un "business model" approprié. On constate que les contractants ne sont pas favorables à une telle utilisation. Si l'ESA l'impose, elle court le risque de voir les industriels se retourner contre elle en cas de problème dans le source libre. Il faudrait préciser les responsabilités, et définir qui assure le support technique et la maintenance des architectures. De plus, il faudrait réfléchir aux moyens d'éviter que le développement libre de départ ne diverge en une multitude de versions indépendantes par industriel.

Le logiciel HYDRA pilote une table vibrante utilisée pour le test de satellites. C'est une application critique d'un point de vue économique, puisqu'une défaillance pourrait entraîner des dommages sur le matériel testé. A l'origine, le logiciel avait été développé selon un processus "chaotique", sans documentation exploitable : il a donc fallu le valider en se basant sur le seul code source. Des premières revues de code ont été conduites pour vérifier le respect de règles de programmation, donnant lieu à une première série de modifications. L'analyse s'est poursuivie par l'identification d'événements redoutés au niveau système. A partir de là, des AEEL et AMDEC, guidées par l'analyse du graphe de contrôle (automatisée par un outil), ont permis d'identifier des défaillances possibles, donnant lieu à des corrections du code source et du manuel d'opération. Au final, le logiciel HYDRA a été amené à un état opérationnel. L'analyse a été réalisée par la société française Surlog. Cette méthode peut-être appliquée à tout logiciel dont on n'a que le source, comme le Logiciel Libre.

Les logiciels libres dans les systèmes sol, *Patrick Pleczon, François Lecouat, Astrium*

Le terme de "système sol" regroupe différents éléments : des moyens pour le test (simulateurs,…), un centre de contrôle satellite, un centre de contrôle lanceur, et des parties relatives à la charge utile des satellites (ex : télécommunications,…).

A titre d'exemple, le système FDC (Flight Dynamics and Commanding System) de Astrium permet de gérer une constellation de 30 satellites (Intelsat). Il possède deux centres de contrôle, sur un site principal et un site back up. Le système FDC est opérationnel 24H/24, avec de fortes contraintes de disponibilité : MTTR de 3mn. Il n'y a pas de temps réel dur, mais une réactivité de l'ordre de 15s est requise. La partie logicielle est en C++ (>100000 lignes de code). Le système FDC s'appuie sur plusieurs COTS : middleware CORBA (Orbix), OS, compilateurs, superviseur de réseaux, générateurs d'IHM, bibliothèque de programmation de contraintes, générateurs de documentation. L'utilisation de tels COTS permet à Astrium de se concentrer sur le développement de produits "métier", et sur l'intégration système. Pour l'instant, il n'y a pas de logiciels libres. L'utilisation de COTS est perçue comme préférable pour disposer d'un support technique contractuel. De plus, si les clients acceptent maintenant le principe d'un système intégrant des COTS, il n'est pas évident qu'ils franchissent le pas pour les logiciels libres.

L'utilisation de COTS dans FDC a cependant posé plusieurs problèmes. Le client a exigé le dépôt des codes sources, qu'il a donc fallu se procurer. Certains COTS se sont avérés bogués, et le problème était parfois bloquant pour Astrium (notamment, un bug système au niveau de l'ordonnancement de tâches a été particulièrement difficile). Rapporter le problème au support technique pouvait nécessiter un effort important, pour isoler une petite partie de code permettant de reproduire la défaillance. Et le support technique du COTS concerné ne s'est pas toujours montré très réactif. Outre les problèmes de versions boguées, on a pu observer des problèmes de cohabitation entre versions de différents COTS (par exemple, une IHM incompatible avec le reste du système). Enfin, le calendrier des versions des différents COTS, avec des évolutions éventuellement divergentes, a été difficile à gérer.

Pour le futur, l'utilisation de quelques logiciels libres dans les centres de contrôle est envisagée, au niveau OS (ex : Linux), middleware (ex : TAO), et au niveau des IHM. Des interrogations subsistent néanmoins quant aux aspects fiabilité, support technique, et juridique. De façon générale, pour les systèmes sol, Astrium n'a que peu d'expérience avec les logiciels libres : quelques cas d'utilisation de Linux, divers utilitaires gnu, et omniORB. Dans ce dernier cas

(simulateurs satellites), si le développement utilisait un ORB libre, la livraison pouvait être faite avec un ORB du commerce, selon le choix du client – d'où une limitation stricte au standard CORBA, pour éviter des problèmes de compatibilité entre ORBs.

3. Prochaines réunions

Le thème "architecture et validation" se poursuivra lors de la réunion 4, avec des interventions de LSV (méthodes formelles), Astrium (architectures bord spatiales), Airbus (architectures embarquées avioniques), et éventuellement LAAS-CNRS (wrappers pour le confinement d'erreurs). J-M. Astruc[2], de Siemens Automotive, sera re-contacté pour un point de vue issu du domaine automobile.

Le plan des 3 réunions suivantes est :

- Réunion 5: processus de création et des évolutions des LL, processus d'utilisation des LL
- Réunion 6: aspects juridiques, certification, protection intellectuelle, sécurité
- Réunion 7: synthèse, préparation de la logique jusqu'à la fin du GT

[2] Il a été invité à participer à la réunion 4.