



*R*éseau
d' *I* ngénierie
de la *S* ûreté de fonctionnement

Développement de systèmes critiques intégrant des logiciels libres

Programme

- Introduction et contexte *Jean Arlat, Philippe David*
- Logiciel libre : de la marginalité à la reconnaissance *Georges Mariano*
- Logiciel libre et systèmes critiques *Philippe David*
- Exemples de domaines d'application
 - ◆ Contraintes et solutions *Philippe Coupoux*
 - ◆ Utilisation de RTEMS dans le spatial *Luc Planche*
 - ◆ Le candidat Linux pour l'avionique embarquée ? *Serge Goiffon*
- Vers une meilleure maîtrise des défaillances *Isabelle Puaut*
- Validation des logiciels libres *Hélène Waeselynck*
- Dimensions juridiques et organisationnelles *Jean-Michel Tanneau*
- Table ronde - Vers une nouvelle orientation au sein des entreprises
Modérateurs : *Alain Costes, Philippe David*
*Pierre-Pascal Galano (Thales), Jean-Max Gaubert (Technicatome),
Famanta Radim (Airbus), Alain Rossignol (Eads-Astrium), Alain Subra (SNCF)*
- Conclusions et perspectives *Jean-Claude Laprie*

Introduction et contexte

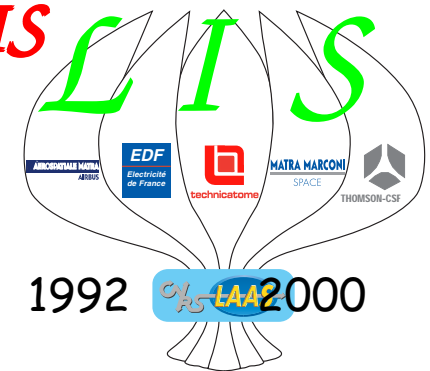


Jean Arlat - LAAS-CNRS

Philippe David - ESA

Le **RIS** : Réseau d'Ingénierie de la Sûreté de fonctionnement

- **Domaine :** Ingénierie de la sûreté de fonctionnement des systèmes à logiciel prépondérant
- Prolongement de la coopération établie dans le cadre du **LIS** sous une forme renouvelée :
 - ◆ Participation industrielle et académique élargie
 - ◆ Favoriser la mise en place de partenariats (programmes de R & D)
 - ◆ Stimuler les réflexions communes
- **Comité d'Orientation** -> Pilotage du réseau
- **Ateliers thématiques** -> Analyse d'un thème
- **Groupes de Travail** -> Réflexion approfondie et document de synthèse







Mise en place en janvier 2001 pour 4 ans

Les activités du *RIS*

(<http://www.ris.prd.fr>)

Ateliers thématiques et groupes de travail

- 1 Logiciel libre et sûreté de fonctionnement  *J. Arlat LAAS-CNRS* LAAS — 14/12/2000
- 2 Usages et perspectives pour la production de logiciels sûrs  *M. Kaâniche LAAS-CNRS* LAAS — 19/10/2001
- 3 Intergiciel et sûreté de fonctionnement  *J.-C. Fabre LAAS-CNRS* LAAS — 6/6/2002
- 4 Nouvelles architecture et technologies des processeurs et sûreté de fonctionnement  *Y. Crouzet LAAS-CNRS*
F. Rodet Technicatome Technicatome — 20/12/2002
- 5 Justification de sûreté de fonctionnement (*dependability case*) : approches industrielles, méthodes de construction et structures *P.-J. Courtois AVN Nuclear*
M. Kaâniche LAAS-CNRS LAAS — 18/3/2003

Logiciel libre : pourquoi ?

- Non viabilité du tout spécifique...
- Intégration de composants **COTS** (*Commercial-Off-The-Shelf*) pas toujours possible...

« Composants logiciels et sûreté de fonctionnement
- Intégration de COTS »



- Alternative ? : **Logiciel libre** (*Open Source*)
 - ◆ Accès au code source -> meilleures observabilité et commandabilité
 - ◆ Communauté de développeurs motivés et compétents -> réactivité et qualité
 - ◆ Application de standards reconnus -> garanties d'interopérabilité
 - ◆ Mais, systèmes critiques soumis à standards rigoureux...

Œuvre collective

■ Rédacteurs

- ◆ P. David *ESA*
- ◆ Y. Crouzet, V. Nicomette et H. Waeselynck *LAAS-CNRS*
- ◆ S. Goiffon *Airbus France*
- ◆ L. Planche *Astrium*
- ◆ G. Mariano *INRETS*
- ◆ I. Puaut *IRISA*
- ◆ B. Bérard *LSV*
- ◆ Y. Garnier *SNCF*
- ◆ P. Coupoux *Technicatome*
- ◆ J.M. Tanneau *Thales*

■ Contributeurs aux réunions

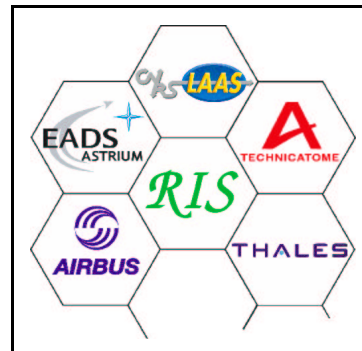
- ◆ J-M. Astruc (*Siemens Automotive*), B. Bray (*Minoru Development*), C. Comar (*ACT Europe*), E. Conquet (*ESA*), J-L. Delamaide (*CEAT*), C. Desquilbet (*CEAT*), R. Dieu (*Airbus France*), A. Fillon (*FIST*), M. Herrb (*LAAS-CNRS*), B. Lang (*INRIA*), G. Lefranc (*Thales*), E. Marsden (*LAAS-CNRS*), Y. Paindaveine (*Commission Européenne*), P. Pleczon (*Astrium*), M. Rodríguez-Moreno (*LAAS-CNRS*), J-L. Terraillon (*ESA*), J. de Urtasun (*Astrium*), T. Vardanega (*ESA*)

Logiciels libres

De la marginalité à la reconnaissance

Georges Mariano

INRETS-ESTAS



Plan général

Origine(s) du libre

Une chronologie simplifiée

Définition d'un logiciel libre

La dialectique des licences

Élaboration d'un logiciel libre

La cathédrale et « les » bazars

Les enjeux du libre

Des logiciels « politiques » ?

Logiciel libre et logiciel critique

Une problématique peu explorée

Repères historiques

1970 – Préhistoire

- ❑ Une diffusion très « libre » des logiciels
- ❑ ARPANET – précurseur de l'internet

1980 – Prise de conscience

- ❑ Entrave croissante aux libertés de l'utilisateur
- ❑ Le manifeste GNU (RMS, 1984)
- ❑ La « Free Software Foundation » (1985)

« Les éditeurs de logiciels cherchent à diviser et à conquérir les utilisateurs, en interdisant à chacun de partager avec les autres utilisateurs [...]. »

« Pour pouvoir continuer à utiliser les ordinateurs en accord avec ma conscience, j'ai décidé de rassembler un ensemble suffisant de logiciels libres, pour pouvoir me débrouiller sans logiciels non libres. »

Richard M. Stallman « Le manifeste GNU » (1984)

Repères historiques (suite)

1990 – Émergence

- Maturité du projet GNU/Linux
- «Open Source», terme déposé par l'OSI

2000 – Reconnaissance par l'intégration

- Dans les stratégies économiques
- Dans les orientations politiques

Définition d'un logiciel libre

Des principes simples

- ❑ Liberté d'utilisation N°1
- ❑ Liberté de modification N°2
- ❑ Liberté de (re)diffusion N°3
- ❑ Liberté d'accès au code N°4

Pour une philosophie paradoxale

- ❑ Contraindre à la liberté

Libre ou pas libre, telle est la licence

Crainte de « l'effet contaminant »

- ❑ Multiplication des licences ad-hoc

Analyse délicate des licences !

- ❑ « Divergences » entre FSF et OSI

Tendance à la «libération» des licences

- ❑ Apple PSL 1.0 (non-libre) / 1.2 / 2.0 (libre)

L'«Open Source selon MicroSoft»

- ❑ Diffusion sous conditions des sources
Windows

Élaboration d'un logiciel libre

Face à la cathédrale...

- ☐ représentant le processus industriel classique

.. « les » bazars

- ☐ une communauté hétéroclite en ébullition

et des fers de lances emblématiques

- ☐ Apache, Gnome&KDE, OpenOffice.org ...

- ☐ La distribution GNU/Linux Debian

Année	1996	1997	1998	2002
Debian	474	974	1500	+10000

Indicateurs de viabilité

L'émergence

- Réponse à un besoin partagé
- Utilisation des technologies standards

Le modèle de développement

- Compétences techniques des développeurs
- Structuration et bénéfice des contributions

Le décollage

- Transparence – Réactivité – Vivacité

Les enjeux du libre

Les limites et les menaces

- ❑ La liberté, un perpétuel combat...

Les législations « périphériques »

- ❑ W indows contre L indows

Les brevets logiciels

- ❑ SCO attaque IBM

Un avenir technologique « mouvementé »

Prise de conscience politique

Niveau local

- ❑ **Contributeurs du libre – Associations (LUG) et collectivités locales**

Niveau national

- ❑ **Fédérations d'Associations – Autorités ou instances gouvernementales (e.g en France, l'ADAE ex ATICA)**

Niveau international

- ❑ **Free Software Foundation – Communauté Européenne (6ème PCRD) – ASEAN**

De la marginalité à la reconnaissance

Reconnaissance économique ?

- Oui, les plus grands acteurs prennent parti

Reconnaissance politique ?

- Oui, toute la « société civile » est impliquée

Reconnaissance technologique ?

- Oui, l'Internet fonctionne sur les principes du libre !

Mais qu'en est-il des domaines critiques ?

- Ultime reconnaissance technologique ?

Logiciel libre et systèmes critiques hérésie ou réalité de demain ?



*Philippe David
European Space Agency*

Premiers constats

- Les fonctions nécessaires aux systèmes critiques sont implémentées par les LL:
 - ◆ exécutifs temps-réel
 - ◆ communication/protocoles
 - ◆ langage
 - Le développement des LL est plus structuré que ce que l'on croit.
 - ◆ Le financement est assuré par des association ou des groupes d'industriels.
 - ◆ Le développement est bien encadré.
- Est-il intéressant d'utiliser les LL dans les systèmes critiques?

Contexte: le futur des systèmes critiques

- La digitalisation de la société rend l'utilisation des systèmes critiques plus répandus
- Contraintes de coût de plus en plus sévères.
 - Privilégier la réutilisation plutôt que de procéder systématiquement à de nouveaux développements
- Interopérabilité des systèmes : système de systèmes
 - Utilisation de standards d'interface (commerciaux ou industriels)
 - Prise en compte des contraintes de sécurité-confidentialité.
- Emploi croissant de logiciels
 - ◆ 48 ko sur SPOT 1 en 1980 → 1,2 Mo sur Mars Express en 2003.
 - ◆ 25 Ko sur A300B en 1974 → 64 Mo sur A380 en 2005.
- Extension de la certification à de plus en plus de systèmes.

Expériences d'utilisation des COTS

■ Besoins du système

- ◆ Connaissance détaillée du COTS
- ◆ Adaptabilité du COTS au système
- ◆ Disponibilité du dossier de certification
- ◆ Disponibilité du COTS sur une période de 5-10 ans
- ◆ Maintenance sur le long terme : 10 à 20 ans
- ◆ Respect des standards
- ◆ Coût et contenu de la licence

■ Inconvénients rencontrés

- ◆ Donner du support peut ne pas être intéressant pour le fournisseur
→ coût du support
- ◆ Le client ne connaît pas le COTS
→ coût du dossier de certification
- ◆ Divergence des intérêts entre l'industriel et le fournisseur sur le suivi du marché → coût de gel
- ◆ Stabilité de la solution dans le temps → coût de maintenance
- ◆ Pour des petites séries → coût des licences est significatif
- ◆ Les clauses de propriété peuvent être bloquantes → négociation des licences

Gestion des risques

- **Risque lié à la licence COTS**
 - ◆ Problème stratégique
 - ◆ Les industriels sont rompus à cette gestion des règles de protection.
- **La défaillance du composant**
 - ◆ Si défaillance, la responsabilité du fournisseur est limitée.
 - ◆ Le risque de propagation des erreurs du COTS vers le système est un risque réel qui doit alors être géré par l'industriel.
 - ◆ L'industriel n'a pas la connaissance du COTS.
 - ◆ La relation de confiance avec le fournisseur est insuffisante à la vue des enjeux pour l'industriel.
- **Disparition fournisseur ou COTS**
- **Risque lié à la licence LL**
 - ◆ Liberté d'utilisation
 - ◆ La GPL apporte des risques nouveaux (contamination)
- **La défaillance du composant**
 - ◆ Utilisateur est seul responsable
 - ◆ Le risque de propagation des erreurs doit être géré par l'industriel.
 - ◆ L'industriel a le code source
 - ◆ L'industriel peut acquérir la technologie
- **LL maintenu par l'industriel**

Impact de la maintenance du système

- **Les durées de vie des systèmes critiques sont longues**
 - ◆ Satellites: 15 ans
 - ◆ Contrôle commande des chaudières nucléaires des sous-marins : 40 ans
- **La politique de maintenance est un des axes de conception du système**
 - ◆ Principes d'architecture qui minimisent les impacts des changements de version
 - Cette conception va favoriser les standards d'interface
 - Des mécanismes d'encapsulation du COTS vont permettre de changer les versions avec un minimum d'impact sur le système.
- **La maintenance long terme doit inclure la gestion du risque fournisseur**
 - Disponibilité du code source est nécessaire

L'apport des logiciels libres pour le système

**COTS → associations d'utilisateurs → standards d'interface
→ LL → Interopérabilité des applications**

- Pas de restrictions d'accès au code source
- L'effort d'appropriation de la technologie d'un LL permet-il de mieux gérer le développement d'un système critique?
- Plusieurs scénarios d'utilisation des LL en fonction:
 1. phase d'acquisition de la technologie du logiciel libre.
 2. effort d'adaptation du logiciel libre au système.
 3. mise en place du dossier de certification.
 4. maintenance en phase opérationnelle.
 5. mise en place d'une équipe de maintenance à long terme du logiciel libre.
 6. gestion des évolutions majeures du système.

Scenario d'utilisation d'un LL

	Acquisition technologique	Adaptation au système	Dossier de certification	Maintenance en opération	Maintenance long terme	Évolutions majeures	Synthèse
Scénario 1 ✓Pas de certification ✓Pas de maintenance	Pas nécessaire	Fournisseur	Non nécessaire	Fournisseur	Code source	Fournisseur	Pas d'investissement. Risque assumé.
Scénario 2 ✓Pas de certification ✓Maintenance	Fournisseur	Fournisseur	Non nécessaire	Par l'industriel	Par l'industriel	Par l'industriel	Investissement dans une équipe de maintenance du LL.
Scénario 3 ✓Certification ✓Pas de Maintenance	Fournisseur	Par l'industriel	Par l'industriel	Fournisseur	Fournisseur	Fournisseur	Acquisition technologique. Certification par l'industriel. Pas de maintenance du LL.
Scénario 4 ✓Certification ✓Maintenance	Fournisseur	Par l'industriel	Par l'industriel	Par l'industriel	Par l'industriel	Par l'industriel	Acquisition technologique. Certification par l'industriel. Équipe de maintenance du LL.

Avantages apportés par les LL

■ Accès au code source

- ◆ Permet de se prémunir contre les risques d'évolution non maîtrisée du logiciel
- ◆ contre le risque de disparition du fournisseur
- ◆ Risque: utiliser le code source en cas de problème sans le maîtriser: c'est le cas aussi pour les COTS

■ Utilisation d'un fournisseur de technologie

- ◆ Même mode de prestation que pour les COTS
- ◆ Le support fourni est souvent de meilleure qualité que pour un COTS car la raison d'être du fournisseur est très clairement sa maîtrise du logiciel

■ Acquisition de la technologie

- ◆ Acquisition de la connaissance fine du logiciel libre peut être longue et coûteuse (plusieurs homme.année)
- ◆ L'investissement est lourd, sur le court et long terme, car il faut conserver une équipe compétente pendant la durée de vie du système.

Sûreté de fonctionnement des LL

- Une structure d'accueil architecturale est nécessaire:
 - ◆ Le logiciel libre est suspect au sens de ses modes de défaillances
 - ◆ Ses fonctionnalités peuvent être surabondantes ou mal adaptées
- Utilisation d'Empaquetage ou wrapper
- Partitionnement du système critique en zones de criticités différentes
 - ◆ Ces mécanismes de confinement des erreurs autorisent l'ouverture des systèmes critiques vers des fonctions d'interopérabilité.
- Sécurité-confidentialité
 - ◆ A prendre en compte car le LL a été développé par un tiers, parfois mal identifié

Certification

- La certification correspond à l'acceptation par un organisme tiers de la preuve que le niveau de sûreté de fonctionnement demandé est atteint.
 - La certification a une forte influence sur la conception du système.
 - Sûreté de fonctionnement et capacité à être certifié ne sont pas prises en compte par les LL.
- Frilosité des industriels à adopter les LL
- ◆ Responsabilité de l'industriel.
- Notre objectif : analyser les processus de certification des divers secteurs industriels pour
 - ◆ identifier les méthodes et efforts à fournir pour rendre le système certifiable
 - LL doit démontrer un apport compétitif certain pour le système
 - Sans introduire de risque supplémentaire

Certification: vue des divers domaines industriels

- la classification en termes de criticité est très homogène entre les divers secteurs industriels.
 - ◆ DAL (Development Assurance Level) dans le domaine aéronautique
 - ◆ ou SIL (Safety Integrity Level) dans le domaine ferroviaire,

Catégorie	Ferroviaire	Aéronautique	Espace	Nucléaire
Pas d'impact	SIL 0	E	/	/
Impact sur le système	SIL 1-2	C-D	critique	B et C
Impact sur les vies humaines	SIL 3-4	A-B	catastrophique	A

Impact des niveaux de sécurité sur l'architecture du système: Aviation Civile

- Analyse de sécurité descendante du niveau système jusqu'à tout équipement terminal qui contribue à la sécurité.
- L'autorité de régulation a dédié un référentiel spécifique, le DO-178B, pour le développement des logiciels de façon à garantir la sécurité du système:
 - ◆ Les commandes de vol électriques, sont mises en œuvre en logiciel.
- Différentes catégories de logiciels (de A à E) sont définies
 - ◆ en fonction de la gravité des conditions de défaillance au niveau système auxquelles le logiciel est susceptible de contribuer.

Solution architecturale à l'utilisation des catégories de logiciel dans aviation civile

Classification des conditions de défaillances	Degré de redondance		
	0 Simple défaillance/erreur	1 Double défaillance/erreur	2 Triple défaillance/erreur
Catastrophique	A	B	C
Dangereux	B	C	D
Majeur	C	D	D
Mineur	D	D	D
Pas d'effet sur la sécurité	E	E	E
DAL logiciel (Development Assurance Level)			

une fonction critique capable d'entraîner une défaillance catastrophique du système peut :

- Ne pas être redondée. Dans ce cas, elle correspondra à un logiciel de catégorie A;
- Être redondée. Chacune des versions du logiciel sera classifiée en logiciel de catégorie B;
- Tripliquée alors chaque version du logiciel sera classifiée en catégorie C.

Certification - LL- Sûreté de fonctionnement

- Un système critique peut être conçu à partir de fonctions moins critiques à condition qu'elles soient redondées et que ces redondances soient gérées selon les exigences de sécurité de niveau système.
- Des protocoles de communication ou des exécutifs temps réel, sont potentiellement utilisables dans les niveaux B ou C.
 - Si redondance alors la certification C ou D est compatible des LL
- L'emploi de logiciels libres aux niveaux A ou B correspond à un processus particulier où le logiciel a été développé spécifiquement par l'industriel et certifié comme tel
 - création de LL

Méthodologie de développement

- **Le DO-178B définit des objectifs à atteindre**
 - ◆ Le contenu du dossier de certification et des éléments de preuve à apporter est un travail de négociation entre l'industriel et le certificateur.
 - **Dans les domaines du nucléaire, ferroviaire et spatial, une méthodologie de développement est imposée par catégorie.**
 - **Trois objectifs fondamentaux :**
 - 1) évitement de fautes apporté par la rigueur du développement,
 - 2) élimination des fautes par les tests et les essais,
 - 3) protection vis-à-vis des fautes résiduelles par l'emploi de fonctions dédiées à la tolérance aux fautes et à l'amélioration de la robustesse du logiciel
- ➔ **Harmonisation aisée entre ces domaines industriels : cycle de vie et des méthodes**

Efforts de développement et test: indépendance par rapport au système

Les efforts à produire sont de deux ordres :

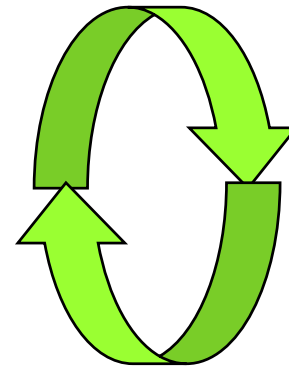
- Des efforts génériques qui ne dépendent que du logiciel et dont les résultats seront exploitables pour tout système hôte qui va utiliser ce logiciel.
 - ◆ Activités de documentation
 - ◆ Tests
 - Des efforts spécifiques au système, qui sont orientés principalement vers la sécurité et l'interface avec le matériel.
 - ◆ Le plan d'assurance de la sécurité. Pour des niveaux SIL 1 et SIL 2 (C, D), les exigences de sécurité sont très limitées.
 - ◆ L'intégration avec le matériel.
 - ☞ le test du logiciel sur le matériel est à refaire pour chaque projet.
- ➔ Les efforts de développement et test du LL sont en majorité indépendants du système ou à refaire de toute façon.

Méthodes applicables à l'intégration de logiciels libres dans un système critique

- L'analyse des différentes normes de certification employées dans les secteurs industriels des systèmes critiques nous a permis de constater
 - ◆ qu'une liste de méthodes communes peuvent être appliquées pour adapter et intégrer le LL dans le système critique.
 - ◆ Ces méthodes dépendent de la criticité du logiciel et peu du secteur industriel concerné.
- Des solutions d'architecture dédiées
 - ◆ Encapsulation
 - ◆ Partitionnement
 - ◆ Mécanismes de protection: Sécurité-confidentialité
- Architecture basée sur des standards d'interface
 - ◆ Favorise les LL
 - ◆ Apporte une bonne durée de vie pour la maintenance

Gains espérés de l'utilisation des LL: le cercle vertueux

- L'utilisation de standards d'interface permettra **d'échanger des logiciels**
- L'effort de certification, qui demande un gros investissement initial pour la constitution du dossier, **peut être partagé.**
- La frilosité des industriels vient de la non compatibilité apparente des LL avec le processus de certification
 - ◆ Nous avons démontré l'inverse avec le support des solutions architecturales
 - ◆ Les risques sont mieux gérés que pour les COTS
 - ◆ Le dossier de certification doit être initialisé par un groupe d'industriels
 - ☞ mise en commun du dossier de certification
 - ☞ Ce dossier pourra être mis à disposition de façon à s'enrichir des expériences d'autres industriels.
- **Amorçage du cercle vertueux**



Conclusion

- l'utilisation de COTS présente des limitations
 - l'emploi de LL apporte de vrais avantages:
 - ◆ respect des standards
 - ◆ moindre risque
 - ◆ code source pour la maintenance
 - Le principal inconvénient
 - ◆ incompatibilité apparente avec le processus de certification
- solutions architecturales
- atelier d'adaptation des LL aux méthodes industriels pour mise à niveau
- Amorçage du cercle vertueux pour une initiative industrielle
- Contribution à la communauté du libre

Exemple de domaines d'application

Quels points d'entrée pour le logiciel libre ?



Contraintes et Solutions

Philippe Coupoux
Technicatome

Évolutions des contraintes (1)

- Les systèmes des différents domaines couverts par les industriels du groupe de travail, spatial, aéronautique, nucléaire et ferroviaire étaient initialement constitués d'équipements dédiés et spécifiques.
- Si la performance technique était présente, les contraintes économiques n'étaient pas la préoccupation majeure des concepteurs pour y aboutir.
- Pour tous ces domaines, les industriels sont passés d'une culture de l'exploit à une culture industrielle, dans laquelle les coûts et les délais de développement sont devenus des paramètres vitaux pour les entreprises.

Évolutions des contraintes (2)

- Parallèlement à l'évolution du contexte économique, qui demande par ailleurs une augmentation de la durée de vie des systèmes,
- La complexité fonctionnelle des systèmes n'a cessé d'évoluer et se traduit par une augmentation continue de la taille des logiciels (de quelques Ko à plusieurs MO aujourd'hui).

Évolutions des contraintes (3)

- Cet accroissement du logiciel doit par ailleurs se faire, dans le respect des documents normatifs de chacun des domaines:
 - ◆ DO-178B pour le spatial et l'aéronautique,
 - ◆ CEI 60880, CEI 1226 pour le nucléaire,
 - ◆ NF EN 50128 pour le ferroviaire
- Ces documents s'intéressent non seulement à la qualité du logiciel final et à ses caractéristiques de conception (déterminisme, contrainte temps réel, tolérance aux fautes...) mais aussi à son élaboration,
- le niveau d'exigences dépend du niveau de criticité de l'équipement.

Réponse des industriels

- En dépit de la diversité des domaines couverts,
- Il existe d'importante similitude dans la conception actuelle des architectures et dans la mise en œuvre du processus de développement, en réponse aux différentes contraintes.
- L'utilisation de composants sur étagère (commerciaux ou non), tend à se généraliser pour les logiciels de criticité faible ou moyenne.
- C'est une tendance motivée à la fois par la réduction des coûts et des délais, mais également par la nécessité de concentrer les équipes de développements sur les produits métier à forte valeur ajoutée.
- Pour les niveaux les plus élevés, cette utilisation semble aujourd'hui non compatible des exigences réglementaires.

Architecture en niveaux (1)

- Sans être révolutionnaire, la mise en œuvre de composant sur étagère (COTS ou LL) s'accompagne généralement d'une architecture logicielle à 3 niveaux :
- Niveau 1 : environnement d'exécution et d'abstraction des ressources matérielles et logicielles,
- Niveau 2 : services standardisés ou propriétaires permettant l'exécution des applications,
- Niveau 3 : Structure d'accueil de haut niveau pour les applications

Architecture en niveaux (2)

- L'utilisation de composants COTS ou LL trouve aujourd'hui une place principalement au niveau 1 (exécutif temps réel, pile de communication bibliothèque d'interface graphique, ...), même si l'atelier de production et de validation semble être un domaine où le logiciel libre fait une percée importante.
- Le niveau 2 est une réponse architecturale à l'intégration de composants COTS ou LL pour en maîtriser l'utilisation, faute de pouvoir en maîtriser complètement la conception.
- Ce niveau permet de limiter la dépendance des applications vis à vis du matériel et donc de les pérenniser, et permet enfin la mise en œuvre de mécanismes de tolérance aux fautes et de confinement d'erreur (à suivre l'exposé sur la technique d'empaquetage).
- Le niveau 3 dépend étroitement du domaine d'application et chaque industriel dispose de réponses architecturales bien adaptées à son contexte.

Conclusion

- Après avoir franchi le pas des COTS, l'utilisation de LL semble être une suite logique, en réponse aux problèmes générés par les COTS.
- Cette utilisation soulève néanmoins de nouvelles questions en terme de maîtrise des défaillances, de validation, de problème juridique et de respect des textes réglementaire.
- A suivre, la mise en œuvre de deux logiciels libres pour le niveau 1:
 - ◆ RTEMS pour le Spatial par Astrium,
 - ◆ Linux pour l'aéronautique par Airbus

Exemple de domaines d'application:

Utilisation de RTEMS dans le spatial



Luc Planche
EADS-Astrium

Contexte

- **Logiciel bord (plate-forme) des satellites d'observation**
- **Développements de logiciels passés/en cours**
 - ◆ Environnement calculateur « 16bits »
 - ◆ Produit « Exécutif temps-réel » maison, limité au besoin, à configuration contrôlée
- **Nouvelle génération de calculateurs « 32 bits »**
 - ◆ Développements en langage C
 - ◆ Nécessité d'identifier un « Exécutif temps-réel » compatible
 - ◆ Choix du produit RTEMS

Solutions compatibles

■ Produit sur étagère

- ◆ Expérience existante sur vxWorks : applications de μ -gravité, à contraintes de sûreté de fonctionnement relâchées et exigeant des services de haut niveau
- ◆ Version 5.3.1 (1997) utilisée sur le système de gestion de données du module européen Columbus, dans le cadre duquel un travail de rétro-ingénierie a été effectué

■ Logiciel Libre

- ◆ RTEMS : Real-Time Executive for Multiprocessor Systems
- ◆ Expérience existante sur RTEMS : application satellite TerraSAR-X (Astrium GmbH)
- ◆ Utilisation de RTEMS encouragée par ESA (de même que pour les environnement Ada, l'ESA encourage le logiciel libre ORK)

■ Développement d'une solution spécifique

- ◆ à l'image de la génération « 16 bits »

Pourquoi RTEMS

■ Solution spécifique écartée

- ◆ Génération de processeurs « 32bits » complexe
- ◆ Expertise requise non rentabilisable dans le domaine spatial

■ Logiciel libre préféré

- ◆ Contexte international (Astrium D/F/UK)
- ◆ Souplesse dans la configuration d'un produit répondant au juste besoin (restriction des services)
- ◆ Souplesse dans l'élaboration et la mise en place d'une politique de maintenance, de pérennisation, et de déploiement
 - ☞ Source disponible, droits de modification & livraison
 - ☞ Pas d'engagement sur l'image d'un fournisseur
- ◆ Mais aussi alignement sur politique ESA (promotion du libre à l'échelle européenne)

Plan de travail

- La version 4.5.0 a été figée et mise en configuration
- 2 axes de travail ont été initialisés
 - ◆ Validation v/v des exigences propres aux applications ciblées
 - ◆ Définition et mise en place d'une politique de maintenance
- Licence
 - ◆ RTEMS est soumis à une licence de type GPL modifiée (licence non contaminante)
 - ◆ Analyse en cours par le service juridique
- Environnement de développement
 - ◆ Utilisation de l'environnement (outils GNU) disponible avec RTEMS
 - ◆ Complété par outils spécifiques pour le développement de logiciel bord

Validation

- Stratégie bâtie sur l'expérience de la génération « 16 bits »
- Il existe un jeu de tests de validation disponible avec le produit
 - ◆ Mais ces tests nécessitent un effort important de rétro-ingénierie
- Couverture structurelle du code source par test (100% des branches exécutées opérationnellement) et analyse (complément)
- Tests spécifiques en boîte blanche des mécanismes de support à l'exécution temps-réel (synchronisation, etc ...)
 - ◆ Les situations jugées pertinentes, de type concurrence d'accès, exécution déterministe, gestion des erreurs etc ... seront identifiées (et définies dans un document de niveau spécification - à confirmer)
- Cette stratégie est en cours de consolidation

Maintenance

- Astrium se propose de maintenir le produit (tout du moins le sous ensemble requis)
- Mise en place d'un processus « produit RTEMS spatial »
 - ◆ Plate-forme d'exécution de référence (autour du processeur ERC32SC)
 - ◆ Gestion en configuration indépendamment des projets utilisateurs
 - ◆ Suivi de la version « publique » (www.rtems.com) pour analyse et prise en compte éventuelle des modifications/évolutions
 - ◆ Gestion des non conformités au cas par cas
 - ◆ Toute modification soumise à l'agrément des projets utilisateurs
- Maintient de la compétence produit par participation aux projets utilisateurs

Exemple de domaines d 'application

Quels points d'entrée pour le logiciel libre ?



Le candidat Linux pour l'avionique embarquée ?

Serge Goiffon (Airbus)

« Tout ce qui n 'est pas donné est perdu. » - Proverbe Soufi

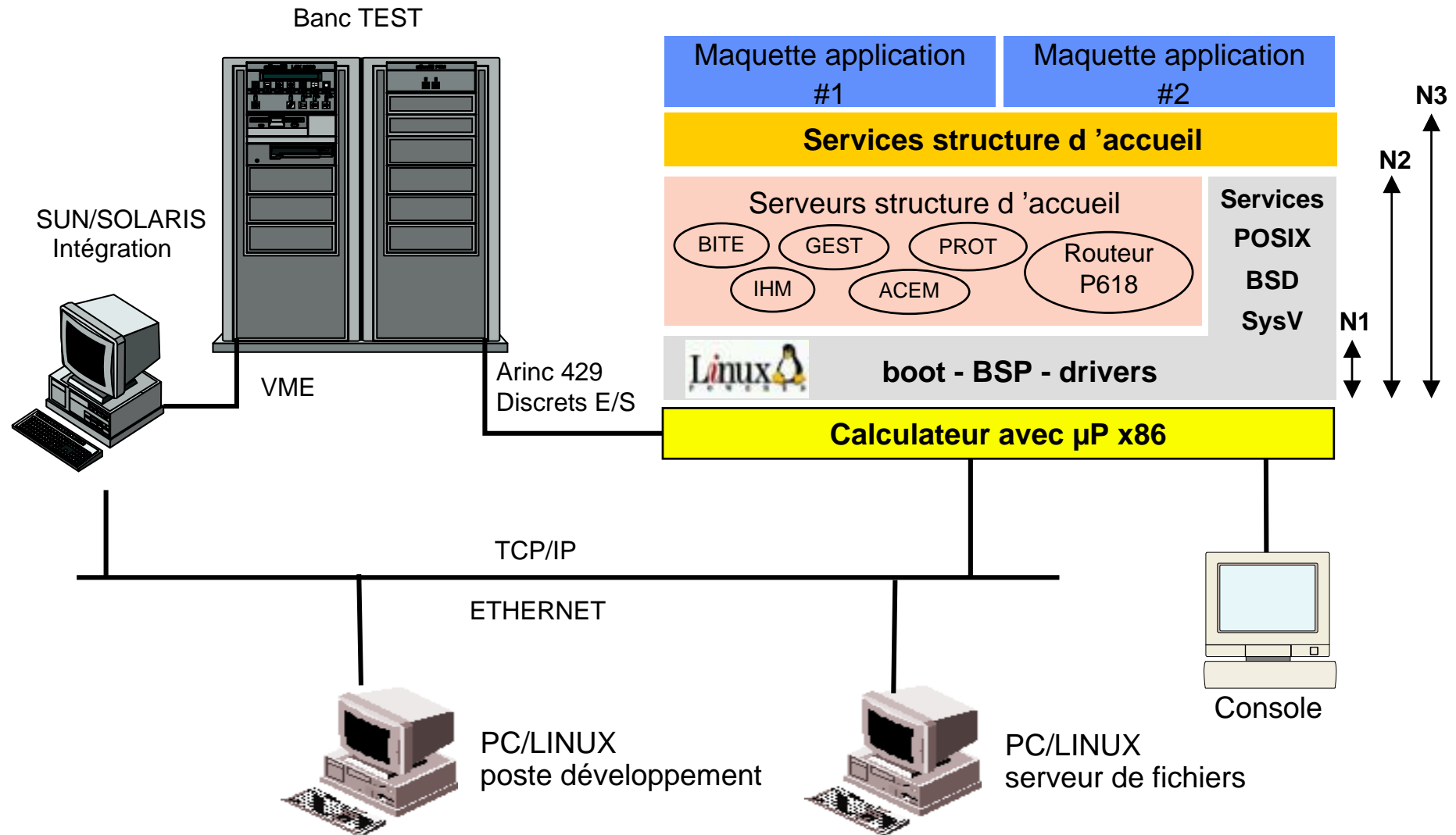
Contexte avionique embarquée

- **Stratégie industrielle basée sur le modèle « tout faire ou acheter »**
 - ◆ **Le « tout faire » n'est plus viable**
 - ☞ Complexité croissante du logiciel embarqué
 - ☞ Composants génériques (O/S, piles de communication, ...) : expertise nécessaire
 - ◆ **L'incorporation de logiciels COTS n'est pas toujours possible**
 - ☞ Objectifs de certification pas toujours partagés par le fournisseur
 - ☞ Dépendance vis-à-vis du fournisseur

- **Le logiciel libre comme solution alternative**
 - ☞ Accès en modification au code source sans redevance
 - ☞ Communauté d'utilisateurs, de mainteneurs, de développeurs
 - ☞ Réutilisation de logiciel, accès à faible coût à la technologie
 - ☞ Modèle de développement coopératif, innovation

- **Questions posées : Linux peut-il ...**
 - ☞ Être embarqué dans du matériel avionique
 - ☞ Être adapté pour répondre aux propriétés avioniques
 - ☞ Entrer dans un processus de certification DO178B

Embarquer LINUX : un ordinateur embarqué sous LINUX



Embarquer LINUX : comment ?

- Base noyau Linux 2.2.12-20 sans patch temps réel
- Adaptations majeures
 - ☞ Code de démarrage et BSP réécrit (pas de BIOS, boot sur un SGF FLASH)
 - ☞ Fonctionnalité XIP ajoutée (exécution code depuis la FLASH)
 - ☞ Universalité des drivers Unix
 - => facile de migrer de LynxOS à Linux (A429, E/S discrète, Flash, EEPROM, Ethernet, ...)
 - ☞ Standard d'interface (POSIX.1, Sys V, BSD, multithreading)
 - => faible impact sur le code applicatif (simple recompilation)
- Validation
 - ☞ Tests existants sur les drivers, OpenGroup POSIX Test Suite, Imbench
 - ☞ Commandes Unix et services réseau
 - ☞ Portage des applications maquettes

Adapter LINUX : contexte avionique nouveau A380

- **Contraintes imposées au niveau du programme avion**
 - ◆ Banalisation ressources de calculs (moins d'éléments de rechange)
 - ◆ Réseau avion basé sur AFDX (Ethernet commuté déterministe)
 - ◆ Norme avionique ARINC 653 comme standard d'interface

- **Propriétés requises**
 - ◆ Partitionnement mémoire
 - ◆ Partitionnement temporel
 - ◆ Partitionnement E/S

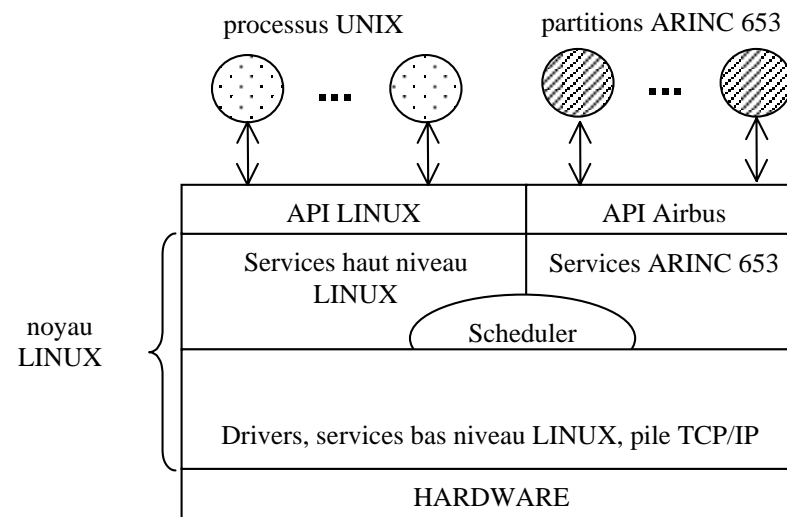
- **A653 => perte d'interopérabilité, nécessite développement d'outils spécifiques (réutilisation difficile)**

- ➔ **Utiliser Linux comme plate-forme d'accueil d'applications avioniques assurant l'interopérabilité et garantissant les contraintes de partitionnement**

Adapter LINUX : la maquette PC LINUX/IMA

■ Modification du noyau Linux 2.4.17

- ☞ 1 partition = 1 processus Unix + contraintes temporelles
- ☞ Modification de l'ordonnanceur (partitionnement temporel)
- ☞ Ajout dans le noyau des services définis par la norme A653 => efficacité par utilisation des services internes Linux pour la synchronisation, allocation mémoire, etc...



■ Adaptation outils

- ☞ Analyse du comportement des applications avioniques : intégration d'événements nouveaux dans l'outil de trace Linux Trace Toolkit

Bilan des études

■ Points forts de Linux

- ☞ Bonne lisibilité du code du noyau, bonne expertise (support de la communauté)
- ☞ Points communs avec LynxOS
- ☞ Interopérabilité (standards industriels)
- ☞ Environnement d'outils GNU complet pour le développement

■ Du point de vue de la certification ...

- ☞ Pas de points techniques bloquants vis-à-vis des contraintes de l'avionique embarquée
- ☞ Nécessité d'un contrôle d'allocation des ressources système et du remplacement des mécanismes non « avionables »
- ☞ Mise en conformité du code vis-à-vis des standards de codage avioniques, peu de document de conception du système, documentation variée, mais incomplète, éparse, et pas forcément à jour
- ☞ Coût d'appropriation non négligeable, nécessite un effort de « reverse engineering » pour recréer le matériel de certification

Vers une meilleure maîtrise des défaillances

L'empaquetage comme moyen de protection



Isabelle Puaut
INSA - IRISA

Problématique

- **Connaissance partielle des caractéristiques du logiciel libre**
 - ◆ Hypothèses de conception
 - ◆ Performances
 - ◆ Propriétés de Sûreté de Fonctionnement : **modes de défaillances**
 - ◆ Fonctions mal documentées
 - ◆ Interopérabilité...
- **Evolutions fréquentes, pas toujours contrôlées**

Nécessité de principes architecturaux pour les intégrer dans des systèmes critiques

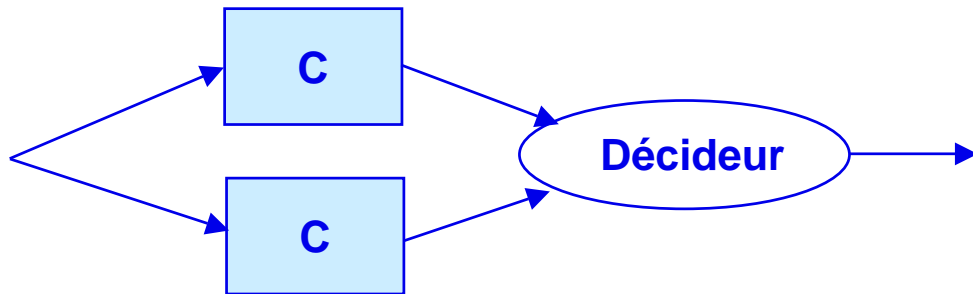
Problématique est similaire à celle d'intégration des COTS

Organisation de l'exposé

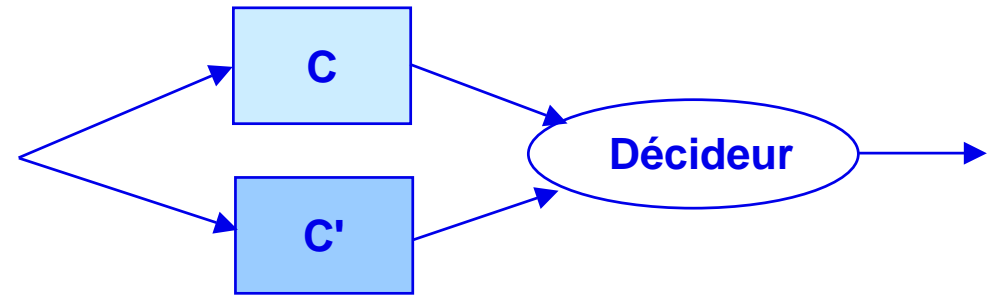
- Solutions architecturales pour l'intégration de LL
- L'empaquetage (wrapping) comme moyen de protection
- Deux expériences d'empaquetage (IRISA - LAAS) :
Empaquetages d'exécutifs
 - ◆ Objectifs de l'empaquetage
 - ◆ Architecture logicielle et contenu des wrappers
 - ◆ Validation des wrappers

Principes architecturaux

Décorrélation de l'activation
(= contextes d'activation différents)



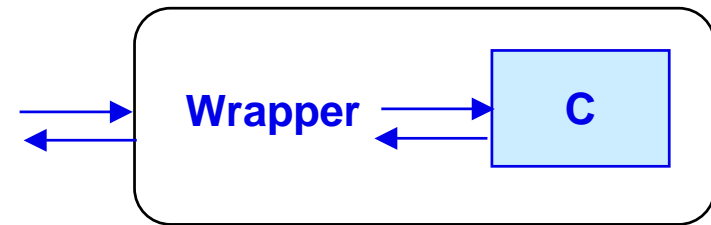
Diversification



Partitionnement

- ◆ Physique
Impact sur architecture matérielle
- ◆ Logique
Spatiale (ex: mémoire),
temporelle (ex: ordonnancement)

Empaquetage



.../...

Empaquetage

■ Origine du terme

- ◆ Groupe de travail de l'ISAT (Information Science and Technology) de la DARPA

■ Deux objectifs

◆ **Adaptation** des interfaces

- ☞ Homogénéisation des interfaces face à l'hétérogénéité des composants intégrés (que l'on veut ne pas modifier dans le cas de LL)

◆ **Protection**

- ☞ Augmentation des capacités de détection et de confinement d'erreurs

■ Intérêt

- ◆ Impact **localisé** sur l'architecture logicielle du système
 - ☞ à l'interface avec les composants utilisés
- ◆ Pas d'impact sur l'architecture matérielle
- ◆ Coût plus faible que les méthodes de diversification

Niveaux de protection

■ Faible

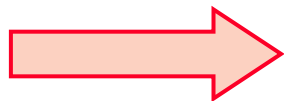
- ◆ Simple adaptation des interfaces (ex: restrictions fonctions API)
- ◆ Complément de fonctionnalités

■ Moyen

- ◆ Vérification des interactions avec le composant (ex: plages de valeurs)

■ Fort

- ◆ Assertions plus poussées sur les fonctions (pré et post-conditions)
- ◆ Très dépendantes de la finesse des spécifications du fonctionnement du composant
- ◆ Très dépendantes du niveau d'observabilité
- ◆ Intérêt dans le cadre des LL : haut niveau d'observabilité et commandabilité (accès au code source)



Le contenu d'un wrapper dépend fortement du degré de protection assuré

Wrappers actifs vs passifs

■ Passif

- ◆ simple notification d'erreur retournée au logiciel applicatif
- ◆ Décision d'engager une procédure de recouvrement est laissée au niveau supérieur

■ Actif

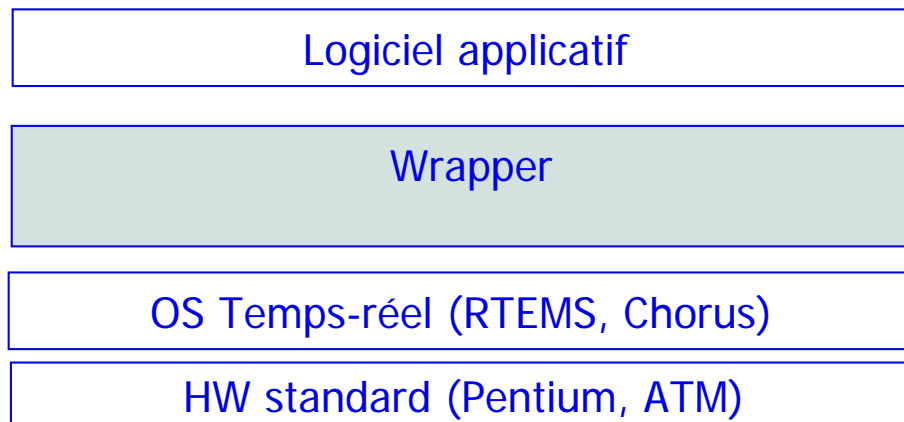
- ◆ Engagement par le wrapper d'une procédure de **recouvrement** (reconfiguration dynamique et reprise du service, re-tentative d'invocation, ...)
- ◆ Rôle **correctif** du comportement anormal du composant encapsulé

Organisation de l'exposé

- Solutions architecturales pour l'intégration de LL
- L'empaquetage (wrapping) comme moyen de protection
- Deux expériences d'empaquetage (IRISA - LAAS) :
Empaquetages d'exécutifs
 - ◆ Objectifs de l'empaquetage
 - ◆ Architecture logicielle et contenu des wrappers
 - ◆ Validation des wrappers

Hades (Highly Available Distributed Embedded Systems) - IRISA

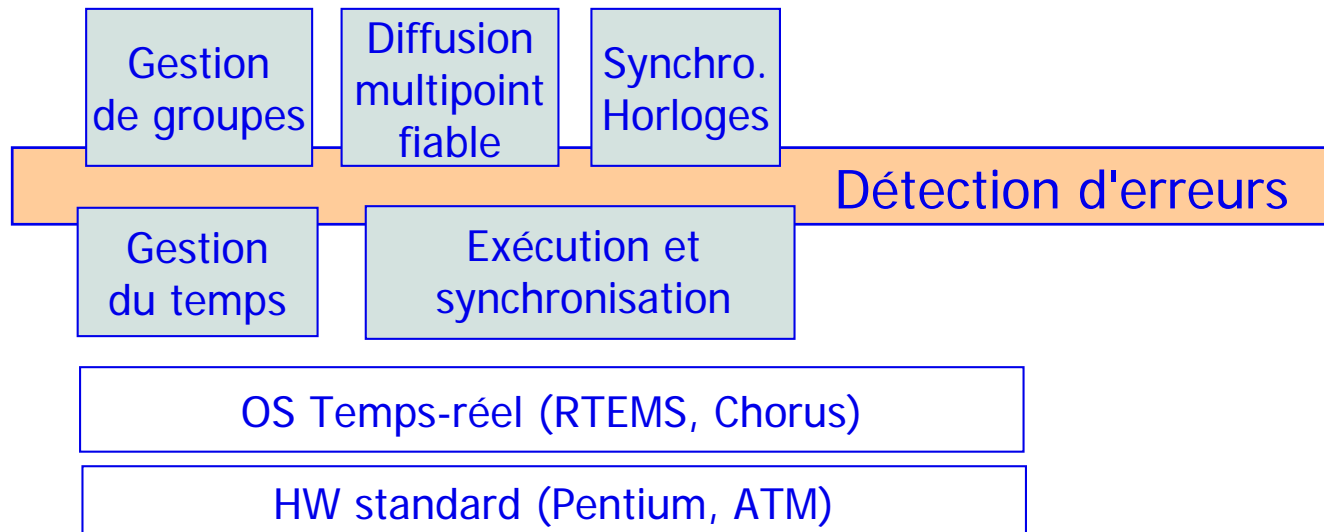
- **Logiciel encapsulé**
 - ◆ exécutif temps-réel (LL ou COTS)
- **Objectif global**
 - ◆ système temps-réel tolérant les fautes physiques, faible coût
- **Architecture**
 - ◆ intergiciel encapsulant l'exécutif temps-réel



**(Coopération INRIA - DGA -
Dassault-Aviation)**

Hades : objectifs de l'empaquetage

- Homogénéisation des interfaces
 - ◆ API indépendante de celle des systèmes d'exploitation encapsulés
- Ajout de fonctionnalités
 - ◆ Gestion de la distribution (synchronisation d'horloges, gestion de groupes, multicast, ordonnancement)
- Détection d'erreurs



Prédicats de détection d'erreurs

■ Types de prédicats

- ◆ Cohérence des structures de données par redondance (ordonnancement, communications, ...)
- ◆ Contrôles temporels (durées d'exécution, échéances, arrivées)
- ◆ Vérifications d'exécution (graphes d'appel, transition d'état des tâches)
- ◆ Vérifications diverses sans redondance

■ Wrapper passif

- ◆ recouvrement d'erreur au niveau supérieur

■ Détection de la propagation des erreurs de l'exécutif vers les wrappers

- ◆ Aucune modification de l'exécutif encapsulé

Evaluation des wrappers

- **Méthode d'évaluation : injection de fautes par logiciel**
 - ◆ Altération de mots mémoires (bit-flip)
 - ◆ Fautes physiques temporaires et permanentes
 - ◆ Cible = exécutif d'une machine
- **Systeme cible : micro-noyau Chorus/ClassiX**
- **Effet des fautes**
 - ◆ Défaillance de l'application (INCORRECT)
résultat incorrect, blocage de l'application, dépassement d'échéance
 - ◆ Détection d'erreur sans propagation aux autres machines (DETECTE)
 - ◆ Application correcte (CORRECT) : erreur masquée ou latente

Evaluation des wrappers

■ Couverture de la détection d'erreurs

	Pas de wrapper	Avec wrapper
Fautes injectées	3152	3012
CORRECT	1839 (58.3%)	1122 (37.3%)
DETECTE, SANS PROPAG	703 (22.3%)	1862 (61.8%)
INCORRECT	610 (19.4%)	28 (0.9%)
Couverture	80.65%	99.07%
Interv. confiance 95%	[79.49 , 81.79]	[98.80 , 99.33]

- Wrappers efficaces : nombre de comportement incorrects divisé par 22
- Redondances entre prédicats de détection d'erreurs

Organisation de l'exposé

- Solutions architecturales pour l'intégration de LL
- L'empaquetage (wrapping) comme moyen de protection
- Deux expériences d'empaquetage (IRISA - LAAS) :
Empaquetages d'exécutifs
 - ◆ Objectifs de l'empaquetage
 - ◆ Architecture logicielle et contenu des wrappers
 - ◆ Validation des wrappers

LAAS - encapsulation d'exécutifs

■ Objectifs

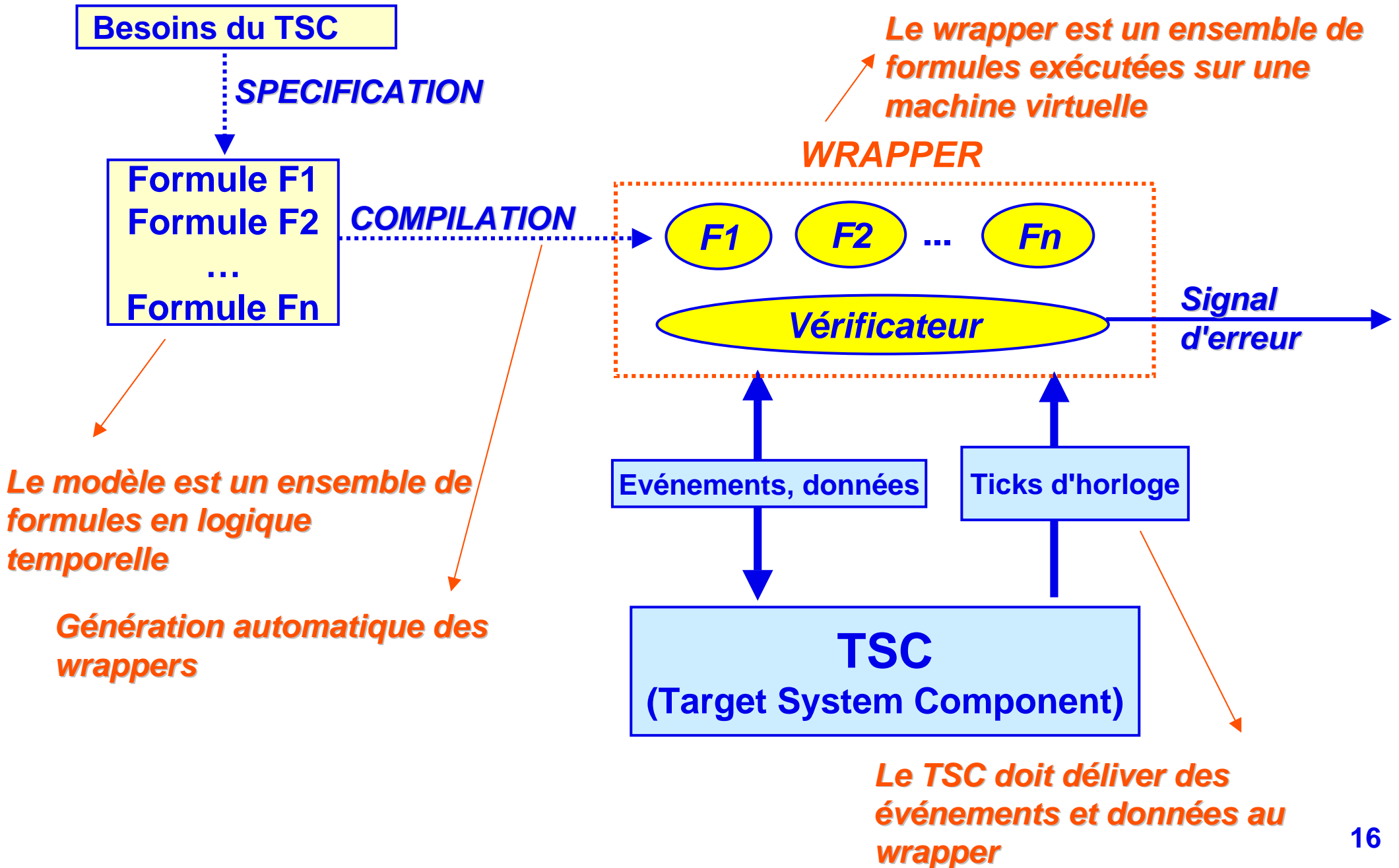
- ◆ Détection d'erreurs : exécution de prédicats élaborés sur le fonctionnement du composant cible
 - ☞ Fautes de conception et de réalisation, effets de fautes physiques
 - ☞ Augmente la sûreté de fonctionnement en évitant la propagation d'erreurs
- ◆ Pas d'adaptation des fonctionnalités ou de filtrage d'API (API inchangée)
- ◆ A la base, détection d'erreurs seulement (wrapper passifs), étendu au recouvrement d'erreur par poursuite (wrappers actifs)

■ Composants cibles

- ◆ Composants arbitraires
- ◆ Evaluation sur les exécutifs temps-réel (suite de l'exposé)

Coopération LAAS-Thales dans le cadre du LIS
Projet IST DSoS

Cadre général



Expression des prédicats

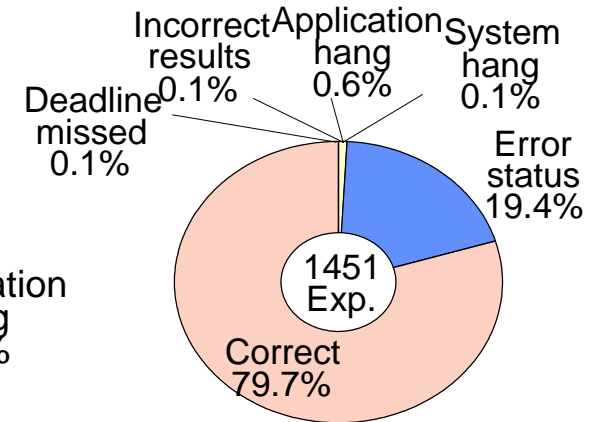
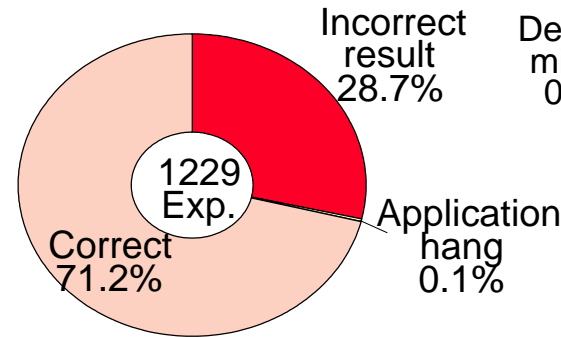
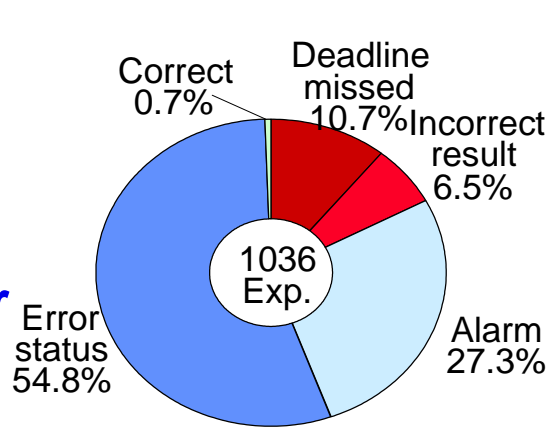
- Types de propriétés classiques : sûreté et vivacité
 - ◆ Sûreté: une situation non désirée ne se produira pas, e.g., interblocage, émission de données incorrectes
 - ◆ Vivacité: une situation attendue se produira, e.g., réponse attendue à une entrée
- Exemples
 - ◆ Propriété de sûreté concernant l'ordonnancement des tâches
La tâche «idle» ne doit pas s'exécuter quand une tâche utilisateur est prête à s'exécuter
 $\text{Always (event=CS} \wedge (\exists s \in \text{Tasks: } s \neq \text{idle} \wedge s \in \text{ReadyQueue}) \Rightarrow \text{Running} \neq \text{idle})$
 - ◆ Propriété de vivacité sur les temporisateurs (timers)
un temporisateur se déclenchera de manière certaine
 $\text{Always (event = SetTimeout (t, tempo)} \Rightarrow \text{Nexttempo (t} \notin \text{TimeoutQueue)})$
- L'expression des propriétés est indépendante de l'implantation du TSC

Evaluation des wrappers

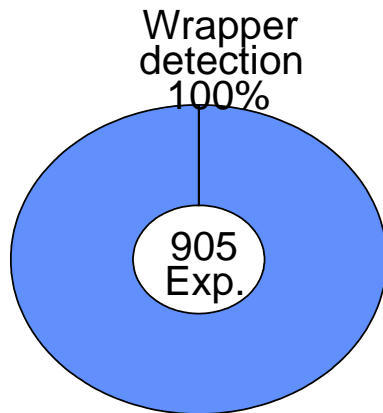
- Méthode d'évaluation : injection de fautes par logiciel MAFALDA-RT (LAAS)
 - ◆ Fautes : d'appel + physiques
 - ◆ Bits-flips dans les paramètres et en mémoire
- Composant cible : micro-noyau temps-réel Chorus/ClassiX
- 31 wrappers dans les modules du noyau
 - ◆ ordonnancement, synchronisation, gestion du temps
- Effets des fautes
 - ◆ Défaillance de l'application (échéance manquée, résultat incorrect, blocage applicatif, blocage système)
 - ◆ Erreur détectée (code d'erreur, alarme)
 - ◆ Erreur masquée ou latente (résultats corrects)

Evaluation des wrappers

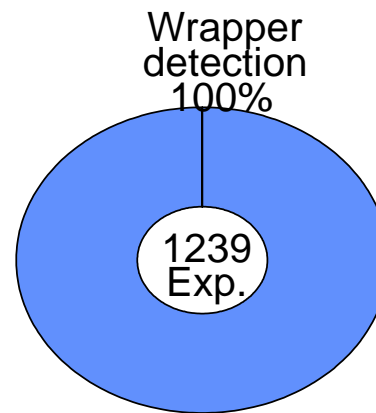
Sans wrapper



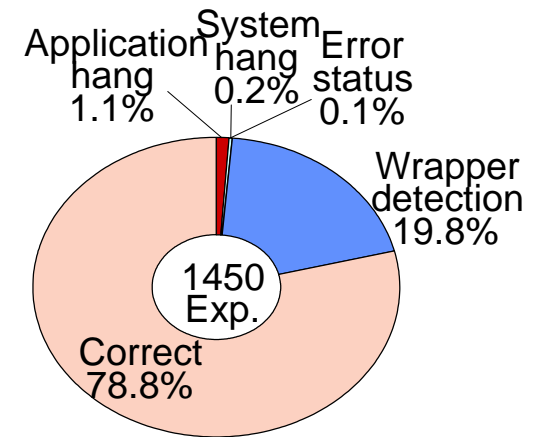
Avec wrapper



Scheduler



Timer



Synchro

- Bonne capacité de détection des erreurs
- Redondance entre wrappers
 - ◆ (4 wrappers détectent toutes les erreurs)

Conclusion

■ Solution générale

- ◆ Adaptation des interfaces faces aux évolutions des LL
- ◆ Détection et confinement des erreurs au niveau du LL intégré
- ◆ Différents niveaux de protection
 - ↳ Niveau de protection élevé atteignable sur des LL (commandabilité et observabilité des LL dû à l'accès au code source)
- ◆ Efficacité de la protection

■ Applicable à de nombreux LL (systèmes d'exploitation - Linux, Corba, ...)

■ Modifications localisées dans l'architecture logicielle du système

■ Coût plus faible que les méthodes de diversification

- ◆ Coût localisé à l'interface des composants encapsulés

Validation des logiciels libres

Mythes et solutions



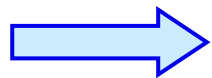
Hélène Waeselynck
LAAS-CNRS

Validation de masse : un mythe

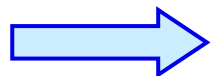
“Given enough eyeballs, all bugs are shallow”

E. Raymond, The Cathedral and the Bazaar

- Croyance que le logiciel a déjà été validé par beaucoup d'autres utilisateurs
- Manque de motivation
- Barrières techniques



Validation: démarche **volontaire** et **organisée**
noyau de développeurs, société support, utilisateur



LL et sûreté de fonctionnement : **l'intégrateur**
doit apporter les démonstrations requises pour
un système cible
focalisation sur le **produit** plutôt que sur le processus

Plan

- Vérification de modèle
- Analyse statique
- Test de conformité
- Test de robustesse

- Aspects spécifiques à la sécurité-confidentialité (malveillances)

- Mutualisation des efforts de validation et capitalisation des résultats

Vérification de modèle (model-checking)

- Modèle = composition d'automates
- Propriété vérifiée = formule logique temporelle
- Idéalement: consolidation de la spécification avant d'aller vers le code
- ... Mais peut aussi être utilisé a posteriori !

Modèle = abstraction du code source

*Approche notamment utilisée
pour la vérification de protocoles*

Protocole de contrôle audio-
vidéo de Bang & Olufsen
(vérification par univ. Aalborg)

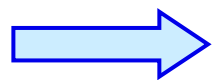
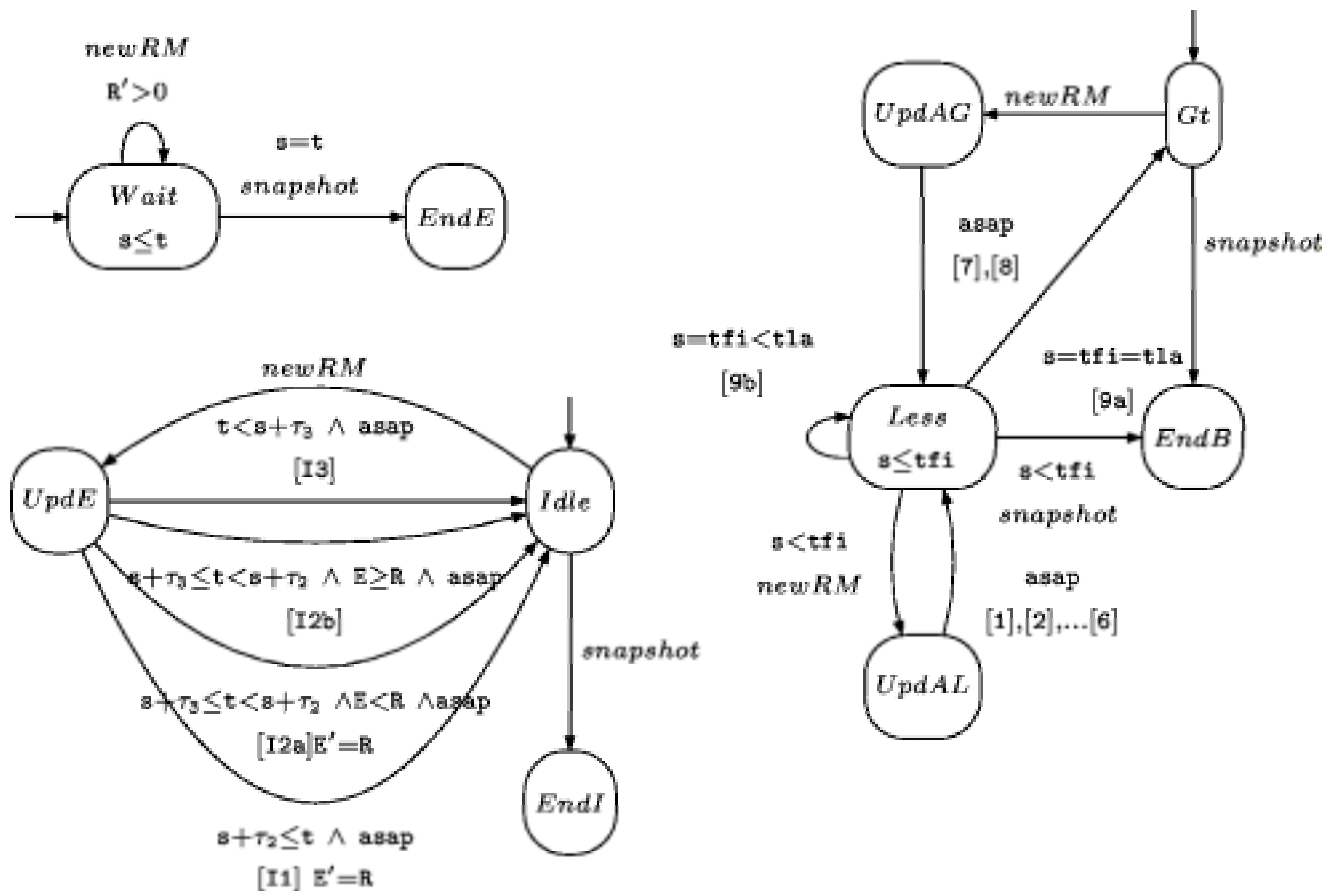
Protocole ABR de France Telecom
(vérification par LSV)
.../...

Algorithme en pseudo-code

```
/* Arrivée d'une nouvelle cellule RM (valeur R) */  
if s < tfi then  
  if Emx <= R then  
    if tfi < s+tau3 then  
      if s+tau3 < tla or tfi=tla then  
        [1] Emx:= R; Ela:= R; tla:= s+tau3  
      else  
        [2] Emx:= R; Ela:= R  
    else  
      if A <= R then  
        [3] Emx:= R; Ela:= R; Efi:= R; tfi:= s+tau3; tla:= s+tau3  
      else  
        [4] Emx:= R; Ela:= R; Efi:= R; tla:= tfi  
    else  
      if R < Ela then  
        [5] Efi:= Emx; Ela:= R; tla:= s+tau2  
      else  
        [6] Efi:= Emx; Ela:= R  
  else  
    if A <= R then  
      [7] Efi:= R; Ela:= R; Emx:= R; tfi:= s+tau3; tla:= s+tau3  
    else  
      [8] Efi:= R; Ela:= R; Emx:= R; tfi:= s+tau2; tla:= s+tau2
```

...

Modèle (LSV)



Vérification d'une propriété de QdS

Analyse statique

- Vérification de propriétés sur le code source, sans exécution de ce code (outils)
- Exemple de propriétés : précision de calculs numériques, débordement de tampons, accès à des variables non initialisées, pire temps d'exécution (WCET), ...



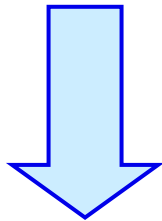
Restrictions sur le code source

Exemple: analyse de WCET sur RTEMS (IRISA)

- Analysé par l'outil HEPTANE (logiciel libre, IRISA)
 - ◆ 14KLOC analysées = 12 directives du noyau (gestion de tâches, sémaphores, gestion du temps, interruptions)
 - ◆ Analyse **code source** ⇒ pire chemin d'exécution
 - ◆ Analyse **code objet + modèle micro-architecture** ⇒ WCET des suites d'instructions correspondantes

Résultats

- Restrictions HEPTANE réalistes pour les sources analysés
 - ◆ Peu de code non structuré
 - ◆ Peu d'appels dynamiques, et fonction toujours connue statiquement
- Identification automatique des portions de code dont le temps d'exécution est non déterministe

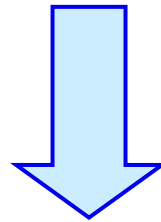


Modifications du code peu nombreuses et généralement mineures

- Obtention de WCET **sûrs** et **raisonnablement pessimistes**
 - ◆ 1.8 x temps mesurés par test
 - ◆ 7 x temps mesurés par test si micro-architecture non prise en compte
- Modélisation incontournable
☺ Intérêt du matériel libre

Test de conformité

- **Vise à vérifier si un logiciel satisfait ses spécifications**
 - ◆ Existence d'un document de spécification ?
 - ◆ Existence d'une documentation de test ? Traçabilité / exigences fonctionnelles ?



- ☹ **Tests fournis (s'ils existent) souvent inexploitable pour validation de conformité**
 - ☹ mais permettent de s'assurer que LL fonctionne comme chez les développeurs
- 😊 **Si LL conforme à standard d'interface (POSIX, CORBA, ...):**
 - ◆ Spécification = document normatif
 - ◆ Existence de tests de conformité (ex: <http://www.opengroup.org>, tests utilisés par Airbus lors des expériences de portage de Linux sur l'ATSU)

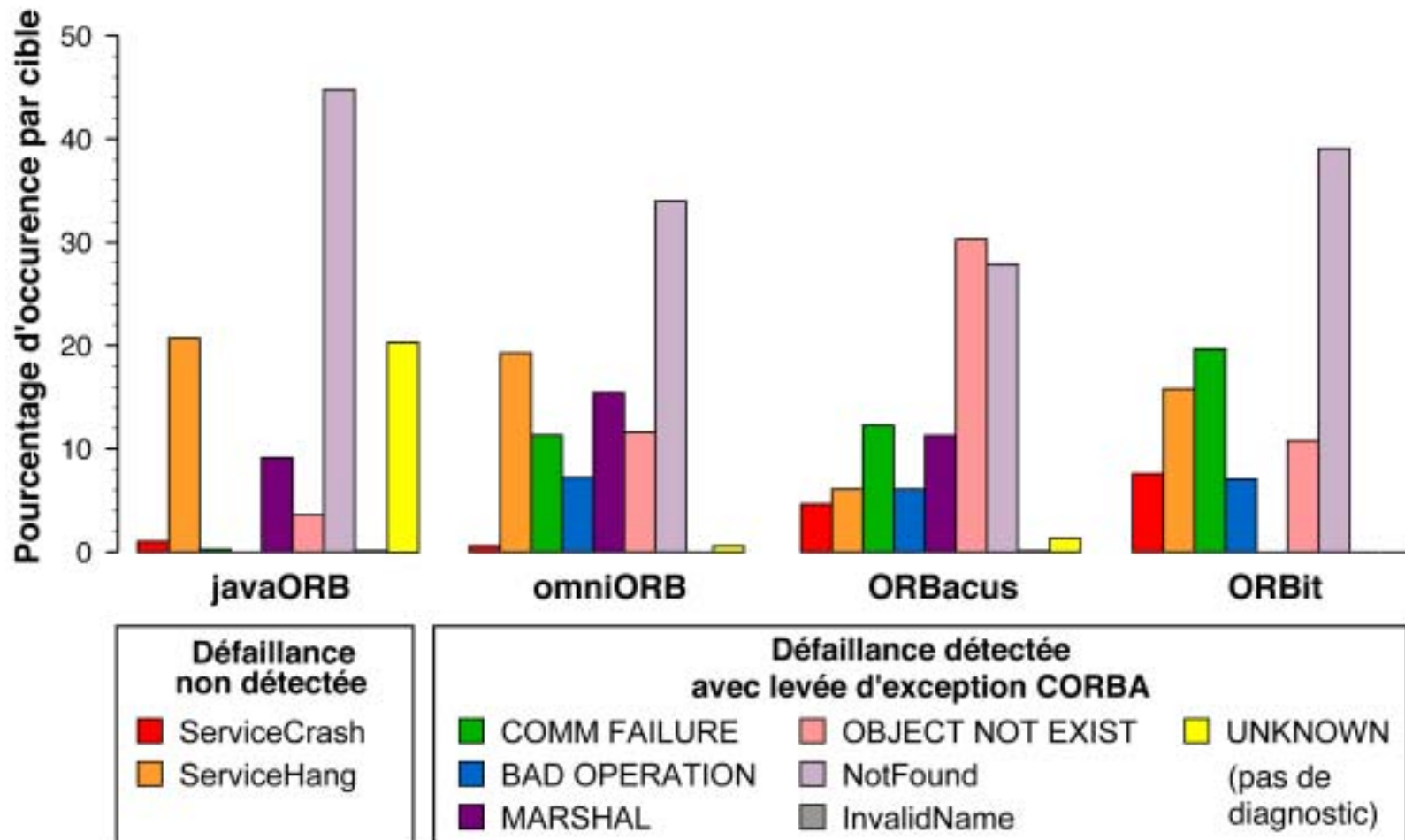
Test de robustesse

- **Test du comportement en présence d'entrées erronées ou de conditions environnementales stressantes**
 - ◆ Comparaison de la robustesse de produits concurrents
 - ◆ Caractérisation des modes de défaillance pour la mise en œuvre de parades (ex: emballage)

- **Approche notamment utilisée sur des supports exécutifs COTS ou LL**
 - ◆ micro-noyaux, systèmes d'exploitation, intergiciels

Exemple: test d'intergiciels CORBA (LAAS-CNRS)

- Test ciblant le « service de noms » de quatre implémentations CORBA (2 COTS + 2 LL)
- Modèle de fautes = corruption de requêtes IIOP entrantes
 - ◆ Bit-flip + double-zéro
 - ◆ Représentatif de fautes transitoires du système de communication (d'après une étude récente [Stone 2000], 1 erreur non détectée toutes les 80 heures sur un LAN 100 Mb/s saturé)



- LL aussi robustes que COTS
 - ◆ les plus robustes : ORBacus (COTS) et omniORB (LL)
- Identification d'une faiblesse commune
 - ◆ Sensibilité à certains bits de l'en-tête IIOP

Aspects spécifiques à la sécurité-confidentialité

- LL plus sûrs (*secure*) que COTS : une vive controverse
 - ◆ source ouvert \Rightarrow pas de logiques malignes et moins de vulnérabilités ?
 - ◆ Mais le source a-t-il réellement été audité en profondeur ? (cf. mythe des "Many eyeballs")
 - ◆ Source ouvert \Rightarrow plus facile pour les attaquants ?
 - ◆ Rq : en pratique, COTS autant attaqués que LL

- Qualité et robustesse du code \Rightarrow impact positif sur la sécurité
 - ◆ Exemples de failles de sécurité dues à des fautes de conception (non-contrôle des débordements de tableaux, des formats de chaînes de caractères, ...)
 - ◆ Pertinence des méthodes de validation présentées précédemment (notamment : analyse statique, test de robustesse / entrées invalides)

Outils spécifiques

/* Identification de vulnérabilités */

- **Analyseurs de vulnérabilités**
État des lieux / vulnérabilités connues
- **Scanners de port**
Identification des services réseaux actifs et de leurs vulnérabilités
- **Générateurs de paquets**
Test de robustesse / faux paquets

/* Aide à la détection d'intrusion */

- **Analyseurs de logs**
Détection du comportement anormal d'un logiciel / système / utilisateur
- **Observateurs du trafic réseau**
- **Vérificateurs de l'intégrité de fichiers**

Mutualisation et capitalisation

■ Ressources offertes par les développeurs

- ◆ FAQ
- ◆ Forums de discussion
- ◆ Rapports de bogues et base de données des bogues

■ Coté utilisateurs : quelques initiatives

- ◆ CERT Coordination Center (DARPA) ⇒ traitement d'incidents de sécurité, base de données de vulnérabilités
- ◆ Projet Sardonix (DARPA) ⇒ portail dédié aux audits de logiciels libres, avec système de notation des auditeurs
- ◆ Linux Test Project (SGI , IBM, OSDL, Bull, Wipro Technologies)

Conclusion

- Qualité de certains logiciels libres comparable à celle de COTS
- Documentation souvent insuffisante pour démonstration de sûreté de fonctionnement
- Mais des solutions de validation existent ...
- ... Et gagneraient à être davantage mutualisées, côté utilisateurs !

Dimensions juridiques et organisationnelles :

quelle stratégie adopter ?

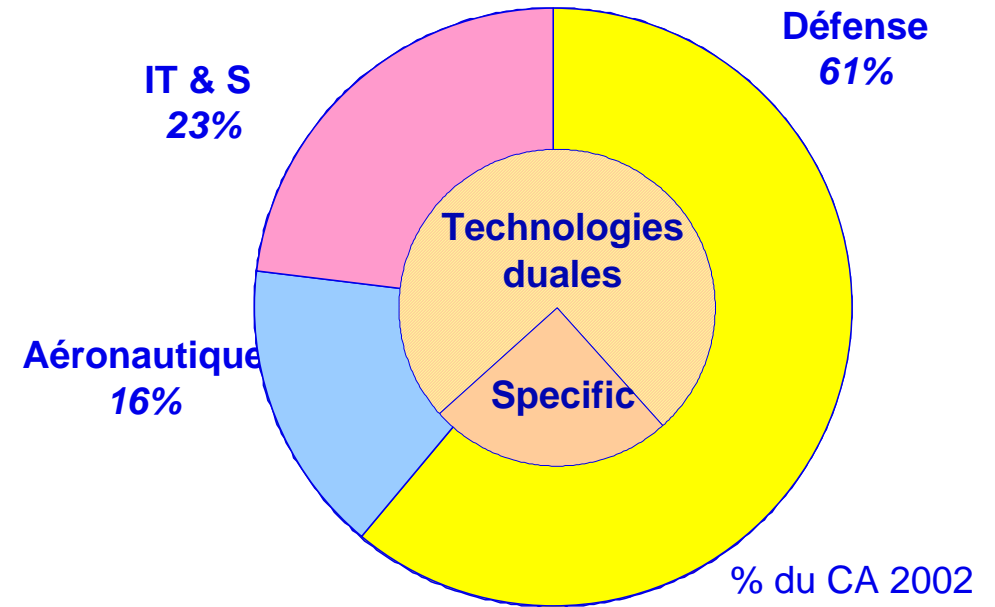


Jean-Michel Tanneau
Thales Research & Technology

Préliminaires

■ Le contexte : Thales

- ◆ Systémier - Intégrateur
- ◆ Multi-domaines
- ◆ Importance des technologies duales (notamment logiciel)
- ◆ Utilisation d'un large éventail des technologies logicielles



■ Une Problématique

Utilisation de LL dans les affaires (Vs infrastructures)

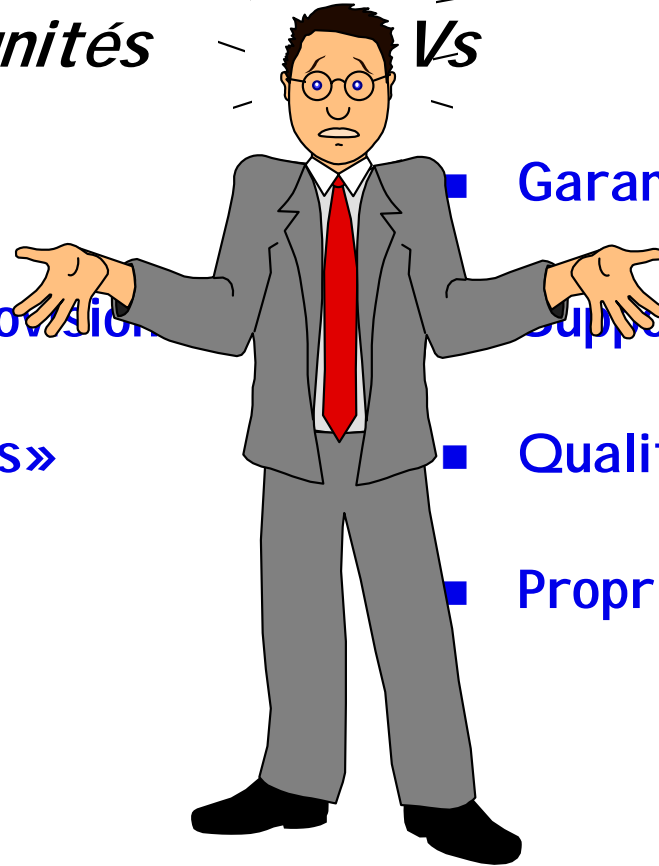
Une première approche

Quels Enjeux ?

Opportunités

- Gratuité
- Autre source d'approvisionnement
- Des logiciels «phares»
- Offre qui s'élargit

Vs

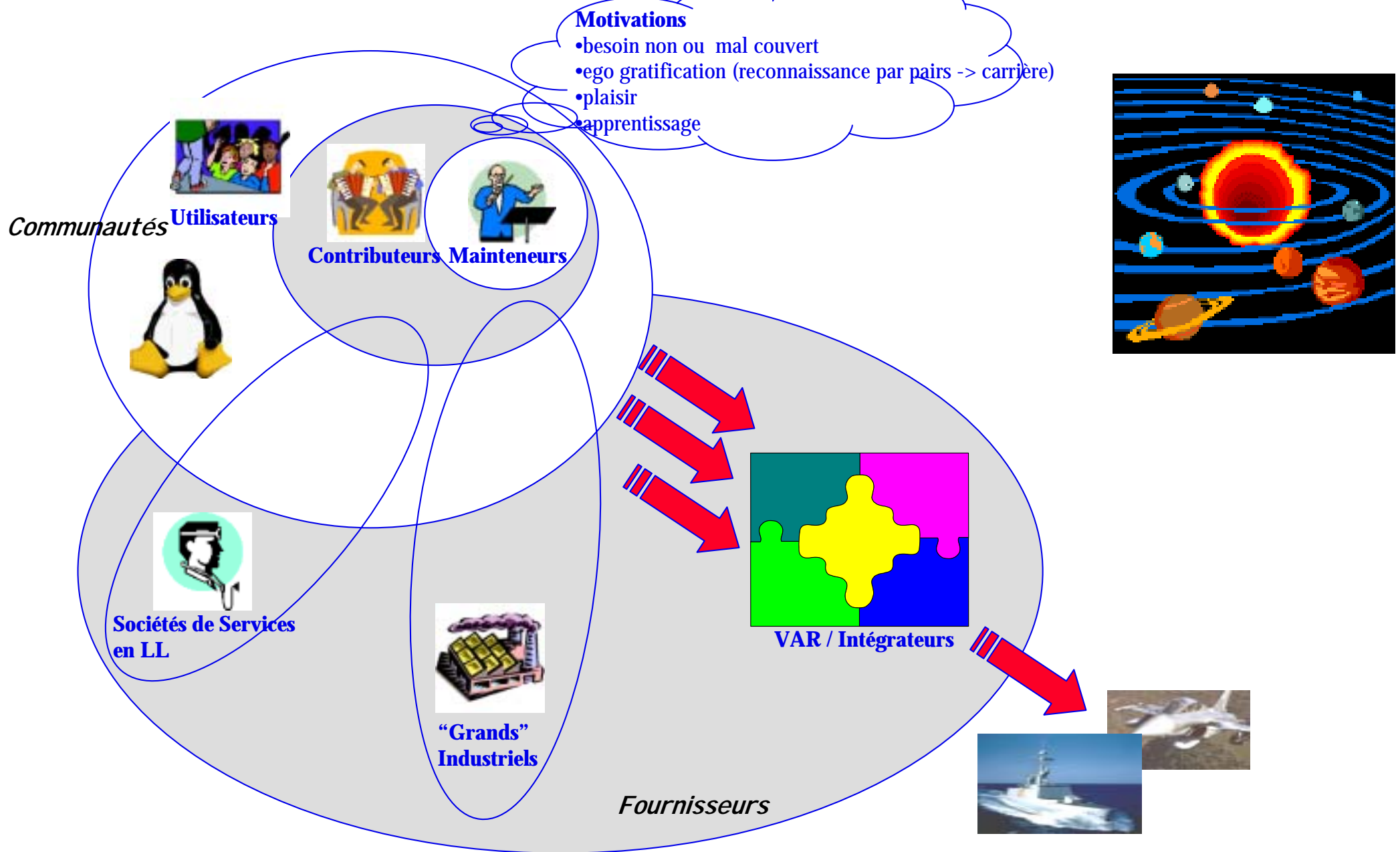


Menaces

- Garantie - Responsabilités
- Support
- Qualité
- Propriété industrielle - FUD

⇒ Analyse comparative LL Vs COTS

Comprendre la galaxie du LL



Un diagnostic : LL Vs propriétaire

Forces

- Mêmes avantages (gain de productivité et de valeur)
- Pérennité - Indépendance fournisseur
- Corrections rapides
- Boîte blanche : Sécurité , Adaptabilité
- Standards
- Support
- Tendances favorables des institutions

Vs

Faiblesses

- Risques sur IPR (licences)
- Évolutions constantes
- Offre diffuse et inégale

Une « Recommandation » pour réduire les risques juridiques

Une gestion de configuration renforcée

Un ajustement du processus évaluation/sélection

Une organisation pour démultiplier les opportunités

Licences de LL et risques

- Licences « virales » (e.g. GNU GPL):
 - ↳ Perte des droits
- Pas de recours en Garantie :
 - ◆ Vices cachés & Défauts
 - ◆ Violation des droits d'une tierce partie
 - ↳ Responsabilité transférée à l'utilisateur

La recommandation

Qualification d'un LL

- ◆ analyse de sa licence
- ◆ recherche de l'existence de brevet tiers le protégeant totalement ou partiellement
- ◆ utilisation documentée
- ◆ existence d'une organisation en charge de l'approvisionner et de le gérer

!!! un LL n'est pas qualifié une fois pour toute !!!

Utilisation dans une affaire : en plus d'être qualifié

- ◆ approbation par Responsable Chargé d'affaire : (contrat, client, législation)
- ◆ Tracabilité par DT unité de l'ensemble des LL utilisés

Évaluation / Sélection de composants logiciels

Objectif

- ◆ Efficacité
- ◆ Confiance (pérennité)
- ◆ Efficience

2 phases

- ◆ Evaluation technique (produits): groupe d 'experts
- ◆ Evaluation « industrielle » (fournisseurs): acheteurs

Évaluation / Sélection

Les débuts

- Objectifs initiaux
- Initiateur
- Origine of the project
- Positionnement
- 1st release stable

Le “décollage”

- Principaux évènements
- Fréquence des releases

Maturité

Comprendre

- Module de développement
- Communauté de développeurs
- Positionnement
- Relation aux standards
- License

Support

- Mailing lists
- FAQ
- Forums
- Sociétés de services

Degré d'acceptance

- Echo dans la presse
- Related web sites
- Industriels impliqués dans le développement
- Nombre de utilisateurs
- Utilisateurs institutionnels & industriels

Futur

- Roadmaps



Comment évaluer?

Google™

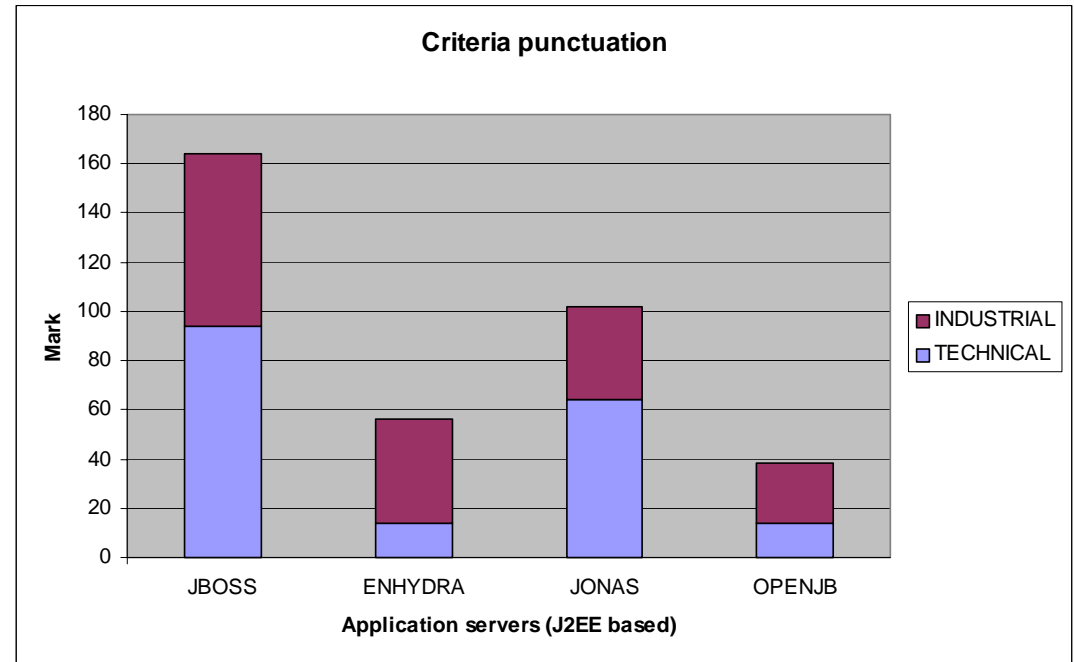
SOURCEFORGE™
net

altavista
THE SEARCH COMPANY

Évaluation / Sélection

<i>Critères "industriels"</i>	<i>Poids</i>
Support technique (SSLL)	20
Base utilisateurs	15
Ré-Activité support communautés	13
Fréquence des releases	12
Société commerciale	10
Nombre de développeurs	8
Forums	7
Documentation	6
Relations avec des LL reconnus	5
Image dans presse/web	4

Total *100*



Première exécution du processus en 2002:

- 326 produits répartis en 43 segments
- publication collaborative portail eCots

En interne - Une organisation dédiée

Un réseau (inter-unités) de correspondants LL

- ◆ Guider et Contrôler (Recommandation)
- ◆ Faire savoir (sensibilisation)
- ◆ Créer et animer des réseaux d'expertise - Veille de l'offre produits serveur d'applications J2EE ???



- ◆ Adapter l'existant (processus d'évaluation/sélection)

- ◆ Veille de l'offre support



- ◆ Capitaliser

- ↳ Analyses de licences
- ↳ Études et Retours d'expériences
- ↳ Utilisations

En externe



■ Promouvoir l'adoption:

- ◆ Le client



- ◆ Organismes de labélisation et/ou financement de la R&D (e.g. ITEA, RNTL)

■ Contribuer et orienter au travers de Consortiums industriels



Conclusion – Démarche d 'appropriation

Utilisation

Adaptation des processus industriels et référentiels

e.g. Evaluation & Sélection, Gestion de l'obsolescence, Guide d'utilisation des LL dans les affaires

D'une utilisation simple (tel que)

à

- une mise en place d 'un atelier** pour
- **robustifier** (pour atteindre un seuil défini)
 - **documenter**
 - **améliorer**
 - **re-ingénierer** *

Outils:

- d'analyse (statique, dynamique, performance)
- de reverse engineering
- de génération automatique de code
- de preuve formelle
- d'injection de fautes
-

* plusieurs solutions peuvent se présenter:

soit convaincre le(s) mainteneur(s) du bien fondé de l'approche

soit *re-prendre la main* sur le projet (avec accord mainteneur (et communautés))

soit être prêt à faire un "fork" (nouveau projet)

Contribution

Développement

(from scratch ou mise en libre d'IP antérieurs)

⇐ Exigences

⇐ Appropriation

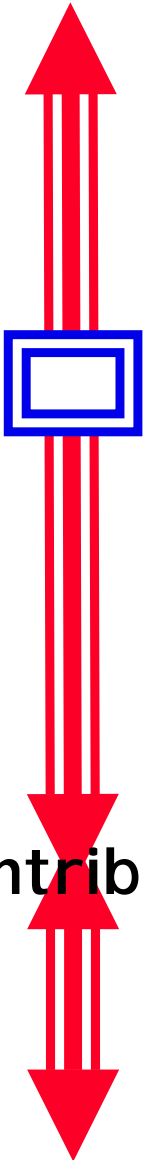


Table ronde:

Vers une nouvelle orientation au sein
des entreprises
quelles propositions et quels enjeux?



Modérateurs :

- ✓ *Alain Costes (LAAS-CNRS)*
- ✓ *Philippe David (ESA)*

Participants :

- ✓ *Pierre-Pascal Galano (Thales)*
- ✓ *Jean-Max Gaubert (Technicatome)*
- ✓ *Famanta Radim (Airbus)*
- ✓ *Alain Rossignol (Eads-Astrium)*
- ✓ *Alain Subra (SNCF)*

Proposition pour l'utilisation de LL dans les systèmes critiques



Philippe David
European Space Agency

Solutions d'architecture à étudier

- L'utilisation des LL est prometteuse et est une tentance lourde
- Les techniques doivent être consolidées pour les introduire dans nos systèmes
 - ◆ L'encapsulation: isoler le LL pour mieux gérer ses versions
 - ◆ L'empaquetage: maîtriser les modes de défaillances du LL
 - ◆ Le partitionnement des fonctions
 - ◆ Les mécanismes de protection: sécurité-confidentialité
- Des solutions complètes d'architecture doivent être définies et qualifiées (famille par niveau de criticité)
 - ◆ Exemple d'une IMA ouverte, en réseau avec d'autres systèmes, redondées, LL (linux, protocoles) au niveau C ou D.

Un processus de développement à formaliser

- **La certification commande des actions**
 - ◆ De développement
 - ◆ D'évaluation
 - ◆ De documentation
 - ◆ De test
- **Il faut définir un atelier de mise à niveau des LL qui soit transverse aux domaines industriels**
 - ◆ Identifier des méthodes communes
 - ◆ Amorcer la dynamique de partage
 - ◆ Mettre les outils correspondants à disposition des volontaires sur un site
- **Initier une structure d'évaluation et de capitalisation des LL**
 - ◆ Validation : caractérisation des modes de défaillances
 - ◆ Mise a disposition des empaquetages
 - ◆ Configuration des LL
 - ◆ Initier la mise en place de dossier de certification en commun

Un processus futur à préparer

- Le logiciel n'est pas le seul support du libre
- LL signifie aussi Hardware Libre (VHDL ou modèles C)
- L'ingénierie système évolue vers la notion de modèle et non plus de logiciel comme l'entité exécutable
- Il faut transcrire cela dans le monde du libre:
 - ◆ Atelier de production système libre
 - ◆ Création de modèles libres

Une nouvelle licence

- Les besoins industriels ou des administrations peuvent être en désaccord avec la GPL.
- Il faut travailler l'idée d'une licence qui facilite l'utilisation de LL dans le milieu industriel
- Initiative au niveau Européen

L'ouvrage peut être commandé à partir du site de Hermes Sciences

<http://www.hermes-science.com/fr/>

Extrait du catalogue thématique « Internet et systèmes d'informations »

Logiciel libre et sûreté de fonctionnement

DAVID P., WAESELYNCK H. (Sous la dir. de)

Env. 256 p., 16 x 24, 2003, relié
ISBN : 2-7462-0727-3, 65 €

nouveauté 2003

Introduire des logiciels libres dans les systèmes critiques semble risqué de par les fortes contraintes qui régissent ces systèmes : exigences de sûreté de fonctionnement, normes de développement drastiques, certification. Toutefois, les logiciels libres sont en pleine expansion et leur usage dans les applications industrielles se confirme, démontrant une véritable évolution de la pratique de l'utilisation des logiciels dans les entreprises. Celles-ci doivent donc s'adapter et trouver des solutions pour rapprocher les pratiques de ce monde du libre des contraintes du domaine industriel.

Les logiciels libres sont, sur certains points, comparables aux logiciels commerciaux sur étagères (COTS) qui sont déjà exploités dans les systèmes critiques. *Logiciel libre et sûreté de fonctionnement* part donc des solutions avancées pour l'utilisation des COTS au sein des systèmes critiques, puis étend l'analyse aux caractéristiques spécifiques des logiciels libres. Il propose des solutions pour gérer les impacts de leur exploitation et pour permettre aux entreprises de ne plus seulement être utilisatrices mais actives dans la création de nouveaux logiciels libres.

Sommaire

Introduction. Repères. Problématique vis-à-vis des systèmes critiques. Besoins et solutions actuelles par domaines d'applications. Expériences autour des logiciels libres. Certification. Validation de logiciels libres. Aspects juridiques et organisationnels. Propositions. Bibliographie / Index.

