État des lieux, et perspectives pour le développement des logiciels embarqués sur satellite

État des lieux

État des lieux (1/3)

- **Les spécifications sont textuelles avec un effort de standardisation**
  L'information du comportemental est transformée par le spécifieur en un format très logiciel, donc entre besoin URD (User Requirements Document) et vue SW, la perte d'information est avérée; l'utilisation de MSC peut être une solution
- **Pour la conception, la méthodologie et les outils HOOD sont abandonnés ; le graphisme HOOD est repris au niveau spécification ou design, il n'est qu'illustratif.**
- **Nécessité du document de conception ? la spécification a une forme objet, orientée logicielle donc contenant déjà le design SW.**
- **La traçabilité via DOORS tend à se généraliser**

© Astrium

État des lieux (2/3)

- **Le développement est effectué sur SUN en ADA avec un soupçon d'assembleur**
- **Les tests unitaires sont toujours menés avec ATTOL. La question de l'utilité des tests unitaires effectués est clairement posée.**
- **Des outils d'analyse statiques, c'est Polyspace qui gagne du terrain ; Logiscope étant moins utilisé**
- **Le programme de production permet de rajouter automatiquement dans le code Ada les structures de données et les constantes qui se trouvent dans la base de donnée système, réduisant ainsi les erreurs humaines de codage et assurant l'homogénéité avec la BD (Référence) au niveau des données).**

© Astrium

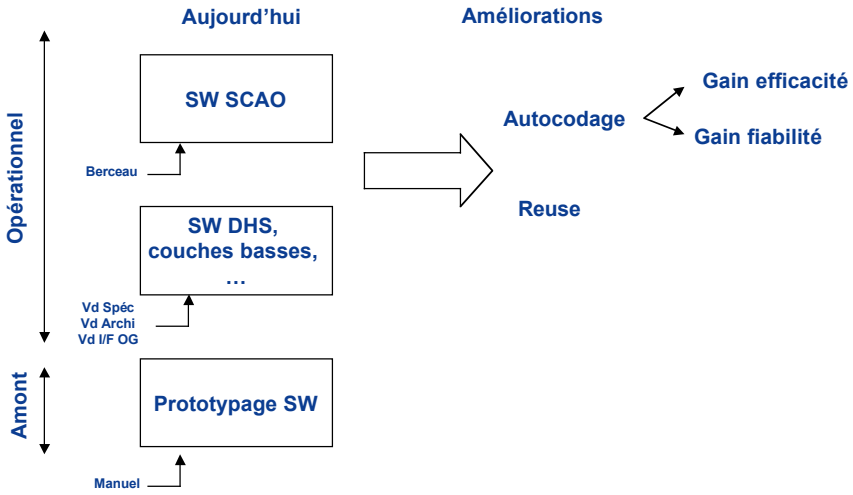
État des lieux (3/3)

- **La génération de l'image mémoire est techniquement compliqué. On trouve des outils de visualisation du mapping mémoire, de génération, de link.**
- **Une réelle standardisation/capitalisation est nécessaire.**
- **La gestion de configuration est faite sur Clearcase.**
- **Pas d'outils pour l'établissement et le suivi**
  - des bilans CPU et mémoire
  - l'observation du temps réel
- **Les idées d'amélioration :**
  - outils standardisés
  - automatiser les activités
  - pérenniser le savoir-faire

© Astrium

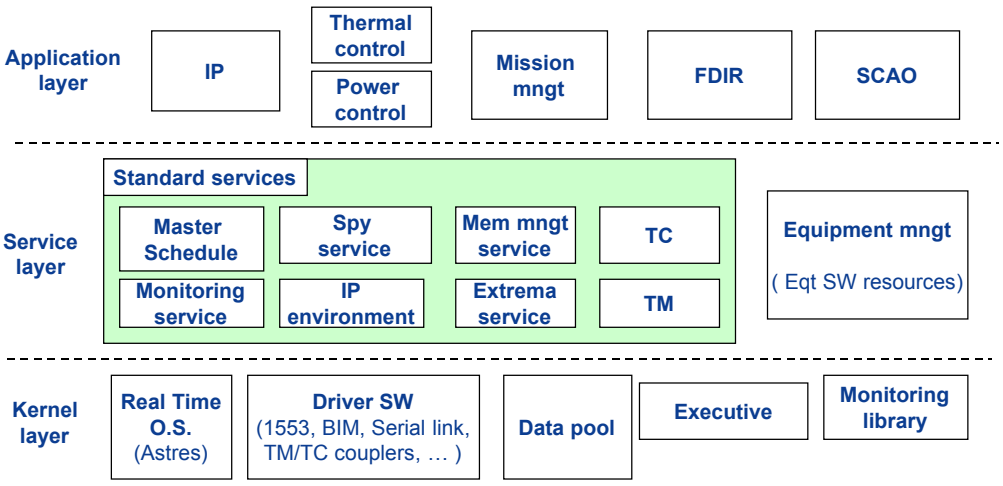
Le process

SW Opérationnel et Amont



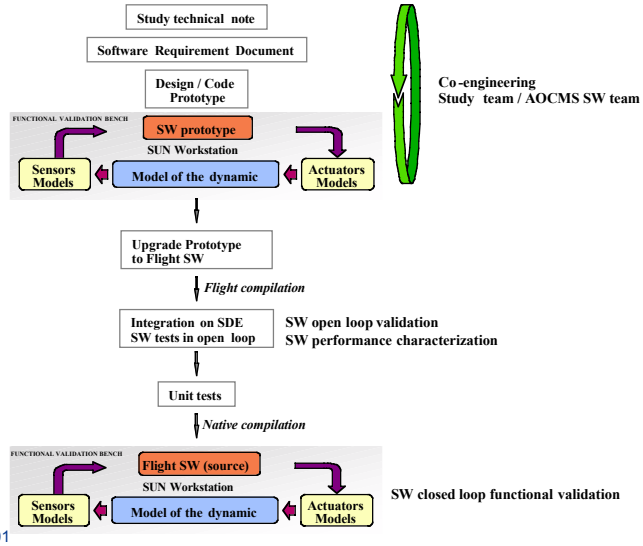
© Astrium

Architecture fonctionnelle générique



© Astrium

Prototypage en équipe intégrée – SW SCAO



© Astrium

Bénéfices de la modélisation

- **La modélisation est une méthode qui permet de gérer les différentes vues du logiciel**
- **Pour formaliser le problème ou la solution:**
  - Un substrat pour la représentation mentale d'un système,
  - Une représentation exécutable: production de traces d'exécution.
- **Pour vérifier:**
  - Le modèle est calculable: preuve de propriétés.
- **Pour communiquer:**
  - Différentes facettes du modèle peuvent servir de support à la communication,
  - Le modèle ne détient pas la vérité mais il permet de dialoguer.
- **Pas d'outil miracle**
  - choix = compromis
  - Couplages a étudier (ObjectGeode/OPNET Pour une étude combinée comportement et performances).

© Astrium

Modélisation pour l'analyse des besoins et spécification

- **Facilité d'expression du besoin.**
  - Langage facile, graphique…
- **Adaptation au problème.**
  - Formalisme naturel pour les spécialistes du domaine.
- **Facilité de communication.**
  - Adaptation à un contexte multi métiers.
- **Vérification des documents.**
  - Cohérence,
  - Complétude,
  - Tests de propriétés,
  - Analyse de complexité

© Astrium

Modélisation pour la construction et validation de l'architecture informatique

- **Facilités de description de l'architecture informatique.**
  - Au niveau système,
  - Au niveau logiciel.
- **Langage simple.**
  - Pour éviter aux utilisateurs de se perdre dans la sémantique.
- **Support à la communication.**
  - Avec le client,
  - Entre les membres de l'équipe projet,
  - Pour informer sur l'état d'avancement.

© Astrium

Modélisation pour la construction et validation de l'architecture informatique (2/2)

- **Vérification et validation.**
  - Pour contrôler au plus tôt le bon fonctionnement,
  - Pour anticiper les problèmes de performances,
  - Pour être sûr que le bon système est en cours de production.

- **Production de documents.**
  - L'approche « tout dans le modèle » est séduisante,
  - Cependant, nécessité de produire des documents papiers,
  - Profiter de la présence du modèle pour produire de la doc.

© Astrium

Phases du cycle de vie : Implémentation et tests de l'architecture

- **Production automatique d'un code.**
  - Pouvoir produire une implémentation à tout moment: approche prototypage permanent,
  - Avoir une stratégie de codage imposée par le générateur: le standard de codage est dans le générateur,
  - Avoir des blocs de bases pré validés que l'on peut réutiliser.
  - S'affranchir des spécificités des plates formes et donc réutiliser des modèles complets.
- **Support à la phase de tests:**
  - Pouvoir générer des séquences automatiquement,
  - Pourvoir les exécuter directement sur cible,
  - Réduction des coûts.

© Astrium

Solutions

- **Très fortes contraintes:**
  - Coût, Planning
  - Augmentation de la complexité de systèmes
- **Deux solutions :**

  La réutilisation: expérience Astrium

  ⇨ La génération automatique de code et de test: au niveau R&D à Astrium

© Astrium

Réutilisation

Les différents niveaux du Reuse SW

On peut distinguer les niveaux de réutilisation suivants avec les éléments logiciels réutilisables associés :

Niveaux	Eléments
Organisation	Equipes
Fonctionnel	spécifications/SRD, architecture fonctionnelle (ex.: archi. fonctionnelle générique)
Architecture	statique (ex. archi. statique générique SCAO), architecture dynamique Temps Réel, interfaces.
Conception détaillée	objets HOOD, code, structures de données
Validation	tests ou types de tests.
Outils / Méthodes	BVL, Attol, Doors, ClearCase, HOOD, SDL, proto AOCS, qualité, process, …
OS, langages	Astres, Ada 83

© Astrium

Comment mettre en oeuvre le Reuse

- **L'expertise technique de réutilisation s'intéresse aux points suivants :**
  - toutes les techniques qui visent à accroître la généricité et l'adaptabilité du logiciel
    - Décomposition du logiciel en couches, développement orienté-objet, techniques de paramétrisation.
    - Identification d'architectures génériques (ex. architecture SCAO Stentor)
  - analyse de la récurrence sur l'ensemble du développement logiciel
    - A un type de composant logiciel correspond généralement des types de tests particuliers.
  - Dépendance de l'architecture logicielle vis à vis de l'architecture matérielle, de l'operating system
    - Architecture centralisée ou distribuée, mise en œuvre de la distribution.
  - les retours d'expérience de réutilisation des projets
    - Confirmation/limite de la réutilisation

© Astrium

La mémoire technique

- **La réutilisation nécessite de conserver une mémoire technique du projet modèle et de la transférer vers les projets réutilisateurs.**
  - Cette mémoire technique couvre tous les domaines du développement, en particulier la justification des choix fonctionnels et techniques.
  - Une rupture de cette mémoire technique peut entamer le bénéfice attendu de la réutilisation.

- **Comment conserver la mémoire technique?**
  => Fiches par thèmes techniques qui formalisent un savoir-faire validé par des experts, fiches projets synthétiques, fiches de synthèse inter-projets.

© Astrium

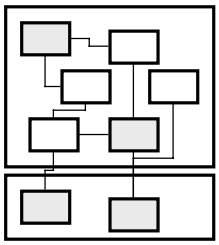
Evaluation et suivi des coûts

- **La récurrence ou réutilisation est souvent surévaluée**
  - L'évaluation est limitée à quelques aspects du développement
  - Ce n'est pas … une solution miracle pour réduire les coûts
  - Les contraintes programmatiques ne sont pas suffisamment prises en compte
  - Les hypothèses de récurrence ne sont pas tenues strictement. Une modification du besoin, mineure au niveau Système, peut entraîner un surcoût significatif.

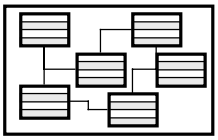
© Astrium

Les pièges du Reuse

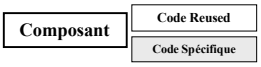
- **Voici deux cas de SW ayant les mêmes « 50% » de Reuse :**



A



B



- **Prédiction des coûts respectifs ?**
  - (B) plus cher que (A) en coût ramenés à la ligne de code
  - => La vision simpliste « %Reuse » est insuffisante

© Astrium

Réutilisation : les problèmes

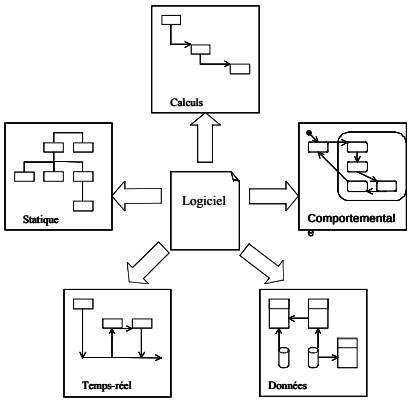
- **Problèmes majeurs**
  - La dépendance des entrées en fonction des projets, même mineure, peut entraîner des surcoûts importants
    - Suite à une modification mineure des besoins, on peut être conduit à particulariser toutes les activités du cycle de développement: modif spec => modif archi => modif code, etc.
    - => effet immédiat sur les coûts : là où on réutilisait en masse, on particularise tout.
    - Choix d 'options par l'ingénierie système, qui font diverger un SW pour une fonction au départ 100% identique.
    - D 'une fonction identique au départ, on particularise toutes les activités du cycle de développement => effet d 'escalier immédiat sur les coûts : là où on réutilisait en masse, on particularise tout avec une obligation de suivi/gestion à chaque étape.
  - Le projet modèle doit être suffisamment avancé afin de fournir une base stable de réutilisation pour les projets réutilisateurs.

© Astrium

Innovation dans le process

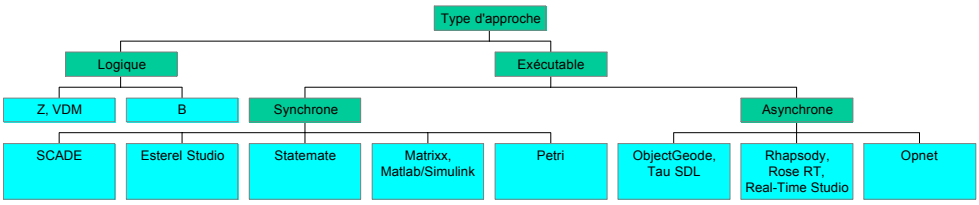
Les différentes vues du logiciel

- **Du fait de sa complexité il faut considérer plusieurs vues d'un logiciel :**
  - hiérarchique (statique)
  - comportementale
  - temps réel
  - données
  - calcul



© Astrium

Panorama des outils



© Astrium

Implication du Logiciel dès les phases A/B

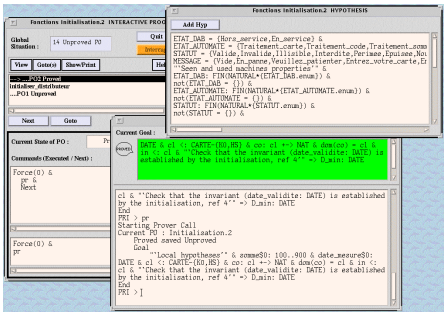
- **Le Logiciel Bord (ingénierie et validation) doit être impliqué dans les phases amont de spécification du Système pour:**
  - Identifier les impacts des deltas au niveau Système sur la récurrence au niveau logiciel.
    - On rappelle encore une fois que de nombreux facteurs peuvent avoir un impact sur la récurrence logicielle : deltas fonctionnels bien sûr mais aussi deltas sur les composants matériels, sur les bancs, sur la BD, etc.
  - Essayer de canaliser les demandes Client et/ou Système en fonction des solutions logicielles disponibles « sur étagère »
  - Pour fiabiliser l'évaluation des coûts de non-récurrence
    - éviter les estimations macroscopiques, prématurées et trop optimistes.

© Astrium

Méthodes et outils
Revue des outils pour sélection

Les approches déclaratives

- Les langages B, Z, VDM utilisent des méthodes fondées sur les mathématiques pour décrire des propriétés ;
- Ces propriétés servent à prédire un comportement par analyse : des théorèmes sont *prouvés* d'après des assertions logiques ;
- Les aspects statiques et dynamiques sont pris en compte par ces méthodes ;
- Mais l'approche est très abstraite et difficilement applicable dans l'industrie.



© Astrium

Les approches « langages exécutables »

- Des représentations graphiques permettent de bâtir des modèles exécutables d'un système temps-réel ;
- Pour vérifier des propriétés, ces approches utilisent des techniques de simulation ;
- Mais la simulation est limitée aux systèmes manipulant des données finies (exemple du lancé de dés) ;
- Il s'agit d'une approche intuitive et très efficace pour valider un système ;
- Deux familles existent : les langages synchrones et asynchrones.

© Astrium

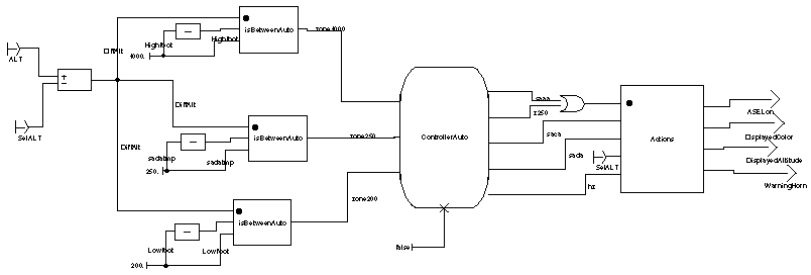
Les « langages exécutables » - Les langages synchrones

- Les approches synchrones font l'hypothèse que l'environnement est lent par rapport à l'exécution du système: hypothèse de synchronisme
- Le fonctionnement du système se fait par pas de calcul ininterruptibles. Toutes les tâches y participent.
- Cette approche permet de garantir des propriétés temps-réel de temps de réaction
- Mais elle ne s'applique pas à la définition de protocoles ou de systèmes répartis
- De plus, l'approche synchrone est difficile à implémenter.
- Adaptés pour modéliser les lois de contrôle, algorithmes de traitement du signal

© Astrium

Les « langages exécutables » - Les langages synchrones - SCADE (Telelogic)

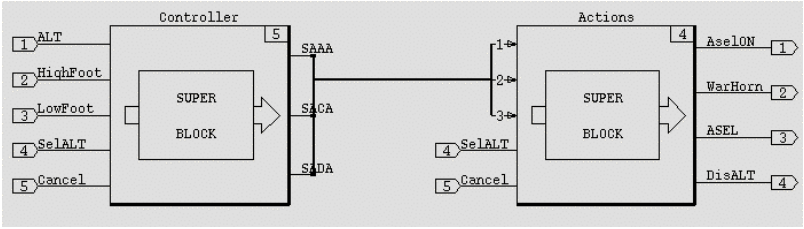
- Editeur graphique s'appuyant sur le langage Lustre (déclaratif) ;
- Modélisation et simulation de systèmes réactifs synchrones ;
- Générateur automatique de code C/Ada certifiable DO178B



© Astrium

Les « langages exécutables » - Les langages synchrones - MatrixX et Matlab

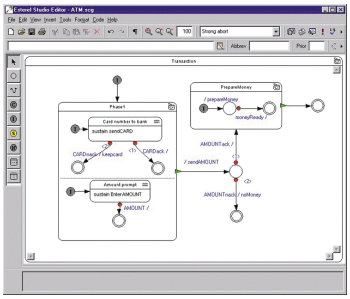
- Outils utilisés pour les études en automatique/traitement du signal
- Modélisation, simulation, génération de code
- Adaptés pour modéliser des lois de contrôle



© Astrium

Les « langages exécutables » - Les langages synchrones - Esterel Studio

- Outil reposant sur le langage synchrone Esterel (INRIA) ;
- Langage impératif pour la spécification/programmation de systèmes TR
- Langage graphique proche des statecharts



© Astrium

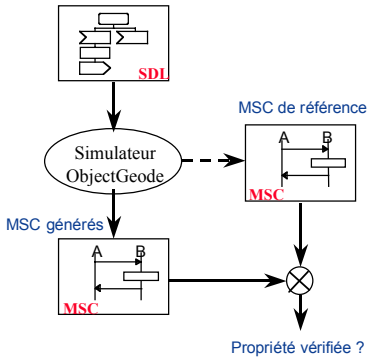
Approches asynchrones

- Les systèmes asynchrones reçoivent des événements de l'environnement;
- La communication entre les tâches se fait par files ou par rendez-vous ;
- Le modèle d'implémentation asynchrone repose sur un système de priorités / politique d'ordonnancement. Il n'y a pas de notion de pas de calcul global.
- Les langages asynchrones permettent de représenter tous types de systèmes ;
- Certains outils permettent d'analyser les performances des systèmes s, asynchrones.
- Adaptés à protocole de communication, traitement de données, traitements répartis

© Astrium

Approches asynchrones - ObjectGeode, Tau SDL (Telelogic)

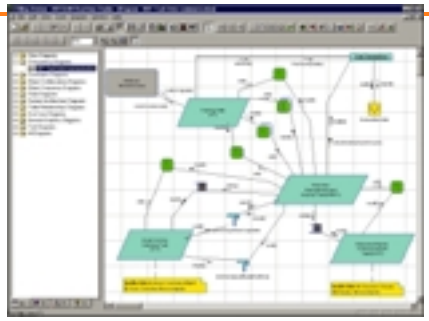
- Outils supportant les langages formels LDS/MSC/ASN.1 (normes ITU-T), et le langage d'observation GOAL ;
- Éditeur graphique avec vérificateur de syntaxe ;
- Simulateur interactif et aléatoire ;
- Vérificateur exhaustif permettant de vérifier des propriétés ;
- Générateur automatique de code ;
- Générateur de séquences de tests ;
- Analyse de performances.

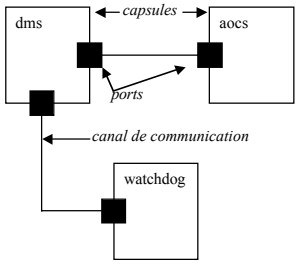


© Astrium

Approches asynchrones - Rhapsody, Rose RT, Real-Time Studio

- Outils supportant la norme UML (Unified modeling language) de l'OMG;
- Extensions pour la modélisation des systèmes temps-réel ;
- Les transitions des automates sont codées en langage d'implémentation ;
- Les possibilités de simulation sont réduites ;
- UML est surtout utile en phase d'analyse.
- Génération automatique de code

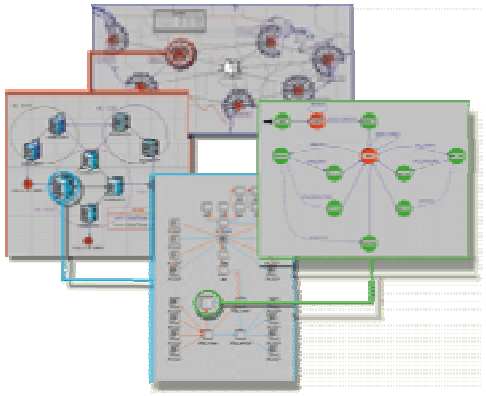




© Astrium

Approches asynchrones - Opnet

- Outil d'analyse et de simulation de réseaux ;
- Éditeur graphique de réseaux, nœuds, processus (statecharts), données ;
- Analyse de performances ;
- Simulateur et outils d'analyse des résultats
- Les aspects analyse de performances sont le point fort de l'outil.



© Astrium

Conclusion sur le panorama

- Il existe de nombreux outils répondant à des besoins différents ;
- Les approches logiques permettent théoriquement de représenter tous types de systèmes mais en pratique elles sont difficiles à utiliser ;
- Les approches exécutables graphiques permettent de modéliser des systèmes synchrones et asynchrones ;
- La simulation exhaustive et l'analyse de performances ne sont pas supportées par tous les outils ;
- Il faut établir des critères afin de sélectionner les outils les plus adaptés aux besoins.

© Astrium

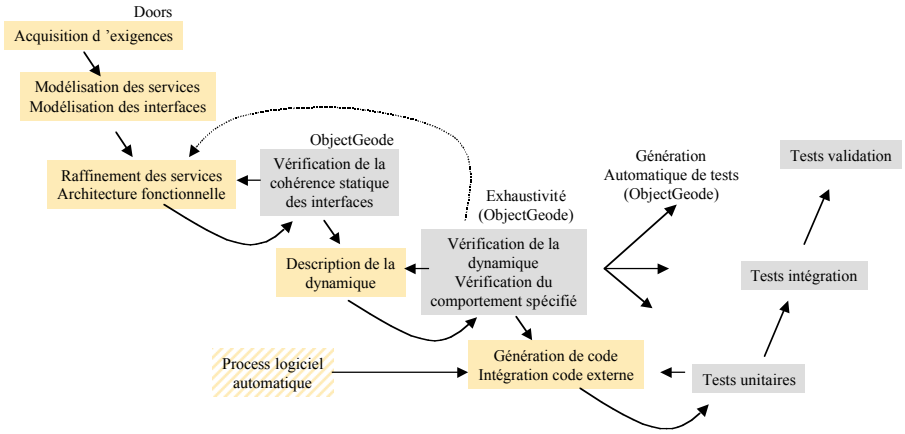
Conclusion

Déploiement de la modélisation dans les processus actuels

- **En phase amont, le logiciel intervient**
  - communications (TC/TM)
  - échanges sur le bus
  - gestion des modes.
  - La modélisation peut participer à la définition statique et dynamique du système, en donnant des premières estimations de charge (traffic) et de combinatoire (cas de pannes, nombre de tests à effectuer).
- **En phase opérationnelle, la mod élisation aide à formaliser la définition de l'architecture du logiciel**
  - données manipulées
  - algorithmes utilis és
  - Comportement
  - performances requises.
  - Il est aussi un moyen d'expression et de capitalisation des données disponibles dans les documents produits à cette étape.
- **Certains outils permettent de générer des scénarios de tests, qui pourront servir, dans les étapes de validation, de référence pour les tests de validation logiciel, voire système.**

© Astrium

Process pour la génération automatique de code et de test – validation incrémentale



© Astrium

Conclusion

- Complexité
- Planning
- Coût

} => Il faut faire une rupture => Cohérence Système SW Test assurée par des outils

© Astrium