

Modèles système, modèles logiciel et modèles de code dans les applications spatiales

ÉRIC CONQUET, FRANÇOIS-XAVIER DORMOY, IULIA DRAGOMIR,
ALAIN LE GUENNEC, DAVID LESENS, PIOTR NIENALTOWSKI
ET IULIAN OBER

Résumé : L'un des principaux risques d'erreurs lors du développement des logiciels embarqués temps réel critiques est la mauvaise interprétation des exigences système allouées au logiciel. Depuis quelques années, l'ingénierie basée sur les modèles réduit ce risque en facilitant la communication entre les ingénieurs système et logiciel. Astrium Space Transportation a ainsi lancé le déploiement de SysML pour la capture des exigences système de la nouvelle version du lanceur Ariane 5 [4]. Au niveau de la conception logicielle, Esterel Technologies commercialise la Suite SCADE [11] permettant la formalisation de la description du logiciel par un langage graphique mixant les automates et les vues flots de données.

Le projet « Full Model-Driven Development for On-Board Software », cofinancé par l'Agence Spatiale Européenne, Astrium Space Transportation, Esterel Technologies, l'Institut de Recherche en Informatique de Toulouse, Altran Praxis et Verimag, a pour objectif de combler les importantes lacunes qui existent toujours entre, premièrement, le modèle système (en SysML) et le modèle logiciel (en SCADE Suite) et, deuxièmement, entre le code généré automatiquement depuis le modèle SCADE Suite et le code manuel (qui n'est pas dans le domaine de modélisation de SCADE Suite). Ce projet traite les aspects suivants :

- Définition d'une sémantique formelle du langage SysML permettant la preuve formelle du modèle système en SysML.
- Intégration d'un modèleur SysML dans la suite SCADE pour permettre une meilleure gestion de la cohérence entre le modèle système et l'architecture du modèle logiciel. Ce travail s'intègre dans le développement d'une nouvelle ligne de produits (SCADE System) développée dans le cadre d'un laboratoire commun avec le CEA (Lis-terel) sur base MDT::Papyrus.
- Génération automatique du code SPARK depuis le modèle logiciel en SCADE.
- Preuve formelle du code final SPARK (automatiquement généré ou manuellement développé).

Une étude de cas développée en rétro-ingénierie du système de génération solaire du cargo spatial ATV validera cette nouvelle approche basée sur les méthodes formelles et les transformations de modèles.

Mots clés : Ingénierie basée sur les modèles, SysML, SCADE, transformation de modèles, génération de code, méthodes formelles, absence d'erreurs à l'exécution

1. INTRODUCTION

De septembre 2004 à décembre 2007, l'Agence Spatiale Européenne (ESA) a dirigé le projet ASSERT [1] du programme FP6 de la Commission Européenne, avec l'objectif de définir un processus complet d'ingénierie système / logiciel pour des systèmes spatiaux complexes et critiques. Le processus de développement d'ASSERT est aujourd'hui assisté par un ensemble d'outils appelé

TASTE [3]. Le domaine principal du projet ASSERT était la gestion des exigences non fonctionnelles, de la capture des exigences système jusqu'à la génération automatique de code. L'objectif du projet « Full Model-Driven Development for On-Board Software » (acronyme FMDE) est de compléter le processus ASSERT par l'approche fonctionnelle en se basant sur les trois piliers suivants :

- • Formalisation de chaque étape du développement par des langages sûrs et non ambigus : SysML pour la conception système, SCADE Suite pour la conception du logiciel et Ada pour le codage
- Utilisation de techniques de preuve formelle permettant d'augmenter fortement la confiance dans le système par rapport à une approche classique reposant uniquement sur des tests :
 - SCADE Suite est déjà un langage formel pour lequel des techniques de preuve sont commercialisées par la société Esterel Technologies
 - Le standard SysML n'est malheureusement pas toujours très précis et laisse une part importante de sa sémantique aux choix des utilisateurs. Le projet FMDE a donc adapté le profil OMEGA (précédemment disponible pour UML) au langage SysML de façon à supprimer toute ambiguïté des modèles et de permettre la preuve formelle
 - Enfin, bien que certainement bien adapté aux logiciels critiques, Ada possède encore quelques constructions ambiguës ou dangereuses. Le projet FMDE a donc proposé d'utiliser le langage SPARK qui, d'une part, restreint le langage Ada aux constructions les plus sûres et, d'autre part, l'étend de manière à permettre la preuve formelle de propriétés.
- Optimisation maximale des étapes successives de raffinement (de la conception du système vers la conception du logiciel puis de la conception du logiciel vers le code d'implémentation) grâce à des techniques de transformation de modèles :
 - Génération automatique du squelette du modèle du logiciel en SCADE Suite à partir du modèle système en SysML.
 - Génération automatique du code de vol en SPARK à partir du modèle logiciel en SCADE Suite par un outil certifié DO178B niveau A.

La Figure 1 résume le processus proposé par FMDE.

La suite de l'article est organisée de la manière suivante. La section 2 présente le profil OMEGA permettant la formalisation du modèle système en SysML. La section 3 montre comment l'architecture du modèle logiciel en SCADE Suite

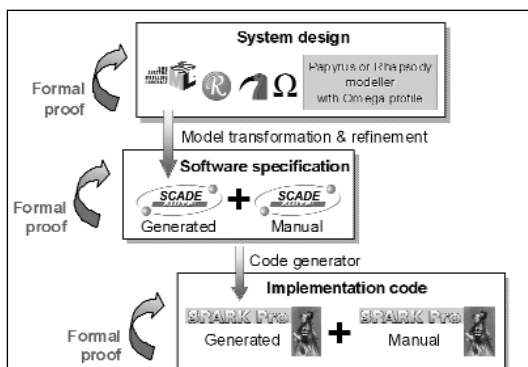


Figure 1 : Un processus d'ingénierie basé sur la modélisation, sur les méthodes formelles et sur la transformation de modèles

peut être automatiquement générée à partir du modèle système en SysML. La section 4 présente le générateur de code SPARK et la section 5 le langage SPARK. Enfin, la section 6 décrit une étude de cas en cours de développement permettant de valider cette approche.

2. CAPTURE DE LA CONCEPTION SYSTÈME AVEC SYSML ET LE PROFIL OMEGA

La modélisation système du projet FMDE s'appuie notamment sur le profil OMEGA [8], profil dédié à la spécification et à la vérification des systèmes temps réel embarqués. Étant à la base un profil exécutable pour un sous-ensemble d'UML, OMEGA a été adapté à SysML. Cela a impliqué l'ajout de nouvelles constructions de modélisation et la définition d'un ensemble de règles pour clarifier certains points de variation sémantique laissés ouverts par la norme SysML. Comme pour la version UML, le profil OMEGA SysML est assisté par la boîte à outils IFx-OMEGA [6].

La composante statique d'un modèle OMEGA SysML met en œuvre des blocs ayant des propriétés, des opérations, des interfaces, des relations de généralisation et d'association ainsi que des signaux. Les diagrammes de blocs internes (IBD) modélisent la structure hiérarchique d'un système. Ceux-ci contiennent des sous-composants avec des ports standard reliés par des connecteurs. Dans le cadre des IBD le profil ajoute un ensemble de règles pour rendre les modèles non ambigus. Ces règles concernent :

- *La direction des ports et des connecteurs.* En SysML les ports sont bidirectionnels, ce qui pose des problèmes de typage. La solution adoptée est d'interdire les ports bidirectionnels et de les remplacer par des ports unidirectionnels. Ne sont acceptés que les connecteurs entre les ports de deux sous-composants ayant des directions opposées et les connecteurs entre un port de l'IBD et un port d'un sous-composant ayant la même direction.
- *Le typage statique des connecteurs.* Les connecteurs partant d'un sous-composant (sans être connectés au moyen d'un port) doivent être typés par une association.
- *Le typage dynamique des connecteurs.* Par défaut, le comportement d'un connecteur est de transférer des requêtes d'une extrémité à l'autre. Pour déterminer les requêtes valides, nous avons introduit la notion d'*ensemble d'interfaces transportées* qui se calcule comme l'intersection des ensembles d'interfaces fournies/requises aux extrémités. En OMEGA, l'ensemble d'interfaces transportées de chaque connecteur doit contenir au moins une interface.
- *Le comportement des ports.* L'ensemble des interfaces d'un port doit être égal à l'union des ensembles d'interfaces transportées de tous les connecteurs partant du port.

► L'ensemble des règles présentées ci-dessus définit un langage bien typé qui a été formalisé en OCL. Plus de détails sur ces principes de bonne formation, leur motivation et leur formalisation se trouvent dans [10].

Le comportement d'un système peut être modélisé en OMEGA SysML par des machines à états et par des opérations invoquant des actions. Le profil OMEGA définit un modèle d'exécution simultanée adapté aux modèles de niveau système et à leur simulation. La granularité de la simultanéité peut être contrôlée par le concepteur du modèle : elle repose sur le partitionnement de l'ensemble des blocs SysML en classes (appelées groupes d'activité) selon l'état actif/passif affecté aux blocs dans le modèle SysML.

En dehors de la spécialisation de la sémantique d'exécution, le profil introduit des extensions pour spécifier le comportement temporisé (temporisateurs et horloges) et les observateurs pour formaliser les exigences et les propriétés dynamiques d'un système. Les observateurs sont modélisés comme des blocs stéréotypés « observer » et définissent une machine à états détectant et rejetant les comportements du système qui ne satisfont pas l'exigence modélisée. Les propriétés exprimées par les observateurs sont toutes des propriétés de sûreté temporisées, basées soit sur l'état du système, soit sur les événements survenant dans l'exécution du système.

La boîte à outils IFx-OMEGA propose des fonctionnalités de simulation et de vérification pour les modèles OMEGA SysML. L'architecture de l'outil s'appuie sur une transformation de modèles OMEGA vers le langage IF [5]. Les règles de transformation peuvent être trouvées dans [9]. L'outil permet de gérer des modèles SysML en XMI 2.1 édités par n'importe quel éditeur SysML 1.1 compatible avec Eclipse EMF comme IBM Rhapsody ou MDT::Papyrus. Le compilateur vérifie l'ensemble des règles présentées précédemment.

3. TRANSFORMATION DU MODÈLE SYSTÈME VERS LE MODÈLE LOGICIEL AVEC SCADA SYSTEM DESIGNER

Comme décrit précédemment, FMDE contribue à la mise en place d'un flot sans rupture et outillé entre les activités d'architecture système et les activités logicielles dans le but d'atteindre trois objectifs principaux :

- Assurer la consistance et la cohérence des activités système et logiciel.
- Éviter les redondances entre activités système et logiciel.
- Faciliter la phase d'intégration par la maîtrise des interfaces et des liens entre les différents composants d'une architecture.

Pour atteindre ces objectifs, après plusieurs années d'études et d'expérimentations, le besoin

de proposer une intégration forte dans un atelier unique d'un modelleur SysML et le modelleur SCADA Suite est apparu incontournable pour complètement répondre aux attentes des industriels.

Dans le cadre du laboratoire commun LISTE-REL, regroupant dans une équipe intégrée des ingénieurs du CEA List et d'Esterel Technologies, une architecture intégrant MDT::Papyrus à l'atelier SCADA a été définie pour constituer le modelleur SCADA System Modeler dans la suite SCADA System. Cette architecture apporte les avantages suivants :

- Les composants MDT::Papyrus partagés avec SCADA System sont *open source* (sous licence EPL).
- SysML et SCADA Suite dans un même atelier avec deux méta-modèles interconnectés permettent de gérer la cohérence et la consistance entre les modèles système et logiciel.
- L'atelier SCADA est maintenant techniquement interopérable avec d'autres composants MDT::Papyrus et ouvert à d'autres modules d'extension Eclipse.
- Une API unifiée (Java ou TCL) est fournie pour accéder en lecture ou modification aux données SCADA Suite et SCADA System (SysML) via leur méta-modèle EMF.
- Un « *look & feel* » unifié est proposé pour minimiser les temps de formation.
- Des moyens de navigation naturels et intuitifs sont proposés entre architectures système et logiciel.
- Des capacités de traçabilité équivalentes et outillées (avec SCADA RM Gateway) pour les aspects système et logiciel permettant la traçabilité entre les exigences et les aspects architecture et logiciel sans rupture.
- Des capacités de production automatique de documents (avec SCADA Reporter) intégrant les aspects système et logiciel relatif à l'architecture.
- Des capacités d'intégration unifiées et outillées avec les outils de gestion de version, configuration ou variante du marché.

Dans la première version de SCADA System mise à disposition dans le projet FMDE les fonctionnalités suivantes ont été intégrées :

- un modelleur SysML intégrant les diagrammes BDD et IBD avec les mêmes fonctionnalités que dans MDT::Papyrus avec une vue arborescente simplifiée et une structure de projet ajoutée,
- un exportateur SysML qui permet d'exporter une partie d'une modélisation système à une autre équipe ou une autre société tout en préservant les propriétés intellectuelles (IP),
- une API SysML qui permet d'accéder au méta-modèle System en s'abstrayant des détails d'implémentation (profils),
- une transformation de modèle SysML vers SCADA qui permet de créer un nœud SCADA ►

TRANSFORMATION

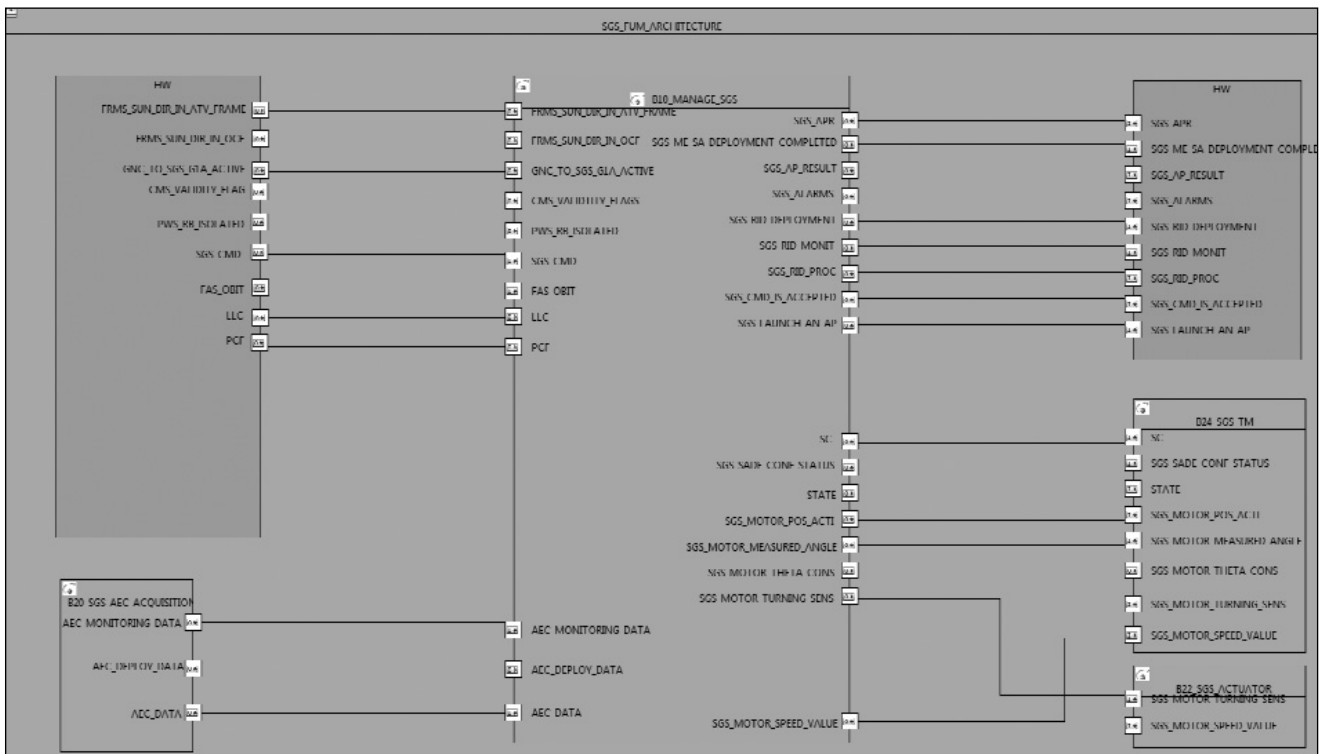


Figure 2 : Exemple de vue Architecture (IBD) de SCADE System

qui possède la même interface qu'un composant (bloc) de l'architecture système avec les mêmes définitions de types (pouvant provenir d'ASN.1) si les types des interfaces sont fournis au niveau système.

4. GÉNÉRATION AUTOMATIQUE DE CODE SPARK DEPUIS LE MODÈLE LOGICIEL

La génération automatique de code à partir d'un modèle SCADE Suite est maintenant une fonctionnalité qui a depuis plusieurs années été intégrée avec succès dans des usages industriels les plus contraints (DO-178B, EN50126, IEC61508). Mais jusqu'à présent, la génération de code C était la seule possibilité offerte dans l'atelier SCADE Suite. Le projet FMDE a contribué au développement d'un générateur de code SPARK pour permettre une meilleure intégration avec un code existant (*legacy*) Ada et qui permet des capacités additionnelles de vérification au niveau du code.

SPARK et SCADE sont deux langages avec des fondations formelles qu'il est naturel de vouloir combiner dans un flot de développement d'applications critiques. De cette combinaison, sont attendus les résultats suivants :

- vérifier la bonne intégration d'une partie de code manuelle dans un code généré automatiquement,
- vérifier au niveau du code l'absence d'erreur à l'exécution (débordement, division par zéro, etc.),
- vérifier au niveau du code des propriétés fonctionnelles.

Pour pouvoir tirer un meilleur profit du langage SPARK, des enrichissements ont été intro-

duit au langage SCADE. Les paragraphes suivants décrivent les principaux enrichissements.

4.1 TYPES NUMÉRIQUES

Nous avons enrichi le système de types avec les types suivants: int8, int16, int32, int64, uint8, uint16, uint32, uint64, float32, float64. Ces types numériques ne sont pas compatibles entre eux, par exemple on ne peut pas additionner un flot de type int8 et un flot de type int32. Pour chaque type numérique, il existe un opérateur « cast » qui permet les conversions de type de façon explicite.

En plus de ces extensions des types numériques, nous proposons d'ajouter un mécanisme pour étendre la hiérarchie de types pour les types utilisateurs importés :

```
type imported T1 is numeric;-
```

- définit un nouveau type numérique (qui est un sous-type de entier ou flottant)

```
type imported T2 is integer ;
```

- définit un nouveau type entier (comme int8/int16/etc.)

```
type imported T3 is floating ;
```

- définit un nouveau type flottant (comme float32/float64/etc.). Les types numériques importés sont incompatibles entre eux et avec les autres types prédéfinis.

Nous introduisons un opérateur « cast » générique (`<expr>:<type_expr>`) pour la conversion de ou vers des types numériques.

Exemple :

```

type imported Timp is numeric;
function F (x1: int8 ; x2:'t)
  returns (y1: float32 ; y2:'t; y3: Timp)
  where 't floating
let
  y1 = (x1: float32); — equivalent to
  float32 (x1);
  y2 = (x1:'t);
  y3 = (x2: Timp);
tel

```

Tous les « casts » entre flottant et entier utilisent la sémantique avec « troncature ». En C, cela était déjà le comportement par défaut. Dans le code généré SPARK, nous utilisons l'attribut 'Truncation avant de convertir une valeur flottante.

4.2 PLAGES DES TYPES NUMÉRIQUES

Il est maintenant possible d'annoter tout type numérique par une définition de la plage de ses limites (bornes maximale et minimale) :

```

int8 { 0 .. 100}
't { -10 .. 10} — where 't integer
T_imp { -3.14 .. 3.14 } — T_imp is floating

```

Nous proposons aussi une nouvelle syntaxe « e1 .. ^ e2 » qui est équivalente à « e1 .. e1+e2-1 » :

```

type T = int { -10 .. ^ 100}; — equivalent to: type T = int {-10 .. 89};
y = x[0 .. ^ 10]; — equivalent to: y = x[0 .. 9];

```

Après ces enrichissements, une correspondance entre certains concepts présents à la fois dans SCADE Suite et SPARK a été réalisée pour avoir un code généré qui soit conforme aux attentes du concepteur. En particulier, concernant les paquets, le générateur de code conservera autant que possible la structure en paquets de SCADE.

5. FORMALISATION ET PREUVE DU CODE D'IMPLEMENTATION AVEC SPARK

Le langage SPARK a été conçu spécialement pour le développement des systèmes critiques où le fonctionnement correct du logiciel est indispensable pour assurer la sûreté et la sécurité du système. SPARK est dérivé d'un sous-ensemble du langage Ada enrichi d'annotations qui facilitent la programmation par contrat [2]. Les outils SPARK (SPARK Toolset) permettent la vérification formelle du code en profondeur et avec une efficacité inégalée. Ils aident aussi à générer la justification de la sûreté fonctionnelle du logiciel qui peut être incorporée dans le dossier de sûreté de fonctionnement requis pour la certification du système. Le langage et les outils ont été utilisés avec succès pour développer plusieurs systèmes critiques dans les secteurs de l'avionique, du transport, du contrôle industriel et de la sécurité.

SPARK repose sur les principes suivants qui le rendent particulièrement bien adapté aux besoins du projet FMDE :

- *Cohérence logique et sécurité fonctionnelle.* La sémantique du code est définie complètement et sans équivoque par son texte et elle ne dépend pas du choix du compilateur. Les insécurités sémantiques présentes dans d'autres langages, telles que les ambiguïtés sur l'ordre d'évaluation ou du mécanisme de passage de paramètres, n'existent pas en SPARK.
- *Simplicité.* Le langage évite toute complexité syntaxique et sémantique, ce qui facilite la compréhension et l'analyse formelle du code.
- *Expressivité.* SPARK offre le support pour l'abstraction, la modularité, la hiérarchisation et les contrats détaillés entre modules, ainsi que pour la programmation temps réel.
- *Vérifiabilité.* Les annotations telles que des pré-conditions, des post-conditions et des invariants de boucle permettent d'exprimer et de vérifier formellement les propriétés désirées sur le code.
- *Correspondance avec Ada.* Tout programme correct SPARK est aussi un programme correct Ada ; il est donc possible d'utiliser n'importe quel compilateur standard Ada pour compiler le code SPARK.

Les outils SPARK sont capables d'analyser les relations entre des sous-programmes et des modules pour découvrir des anomalies dans le code. L'analyse du flux de données assure l'initialisation correcte de toute variable avant son premier usage et détecte tout usage (et non-usage) des paramètres qui n'est pas compatible avec leur mode. L'analyse du flux d'information considère les dépendances entre les données d'entrée et de sortie des procédures et des fonctions et, grâce au raisonnement compositionnel, d'autres procédures, fonctions et modules qui les utilisent. Cette analyse permet aussi de détecter des boucles infinies.

La profondeur et la complexité de la preuve formelle d'un programme SPARK dépendent du niveau d'intégrité du système et du niveau souhaité de certification. Dans sa forme la plus simple, la preuve consiste à démontrer l'absence d'erreurs à l'exécution, c'est-à-dire à prouver formellement et avant que le code ne soit exécuté qu'il n'y aura pas d'erreurs telles que la division par zéro, l'utilisation d'un indice en dehors des limites d'un tableau d'éléments ou bien l'affectation d'une valeur trop grande ou trop petite à une variable d'un certain type. La forme la plus complète de la preuve, si on dispose d'une spécification formelle et détaillée, est de démontrer que le code satisfait cette spécification. Dans la majorité des cas, on choisit une combinaison de ces deux méthodes : une preuve de l'absence d'erreurs à l'exécution est effectuée sur l'ensemble du code et couplée à une preuve complète des propriétés désirées des com-

► posants les plus critiques. Cette approche mixte sera aussi utilisée dans le cadre du projet FMDE. Nous allons vérifier l'absence d'erreurs à l'exécution dans l'ensemble du code SPARK généré à partir du modèle SCADE ainsi que dans la partie algorithmique écrite à la main. Nous allons ensuite démontrer que certaines propriétés spécifiées initialement au niveau système sont préservées par le code exécutable.

6. VALIDATION DE L'APPROCHE SUR UNE ÉTUDE DE CAS

L'approche FMDE, basée sur la modélisation, les méthodes formelles et la transformation de modèles, est en cours de validation sur une étude de cas tirée du programme ATV (Automated Transfer Vehicle). L'ATV, développé sous maîtrise d'œuvre Astrium Space Transportation pour le compte de l'Agence Spatiale Européenne, est un cargo spatial mis en orbite par le lanceur lourd européen Ariane 5 et ravitaillant la station spatiale internationale (ISS). L'ATV est le système le plus complexe et le plus critique jamais lancé par l'Europe de l'Espace. L'étude de cas consiste plus particulièrement en la gestion des panneaux solaires du véhicule qui lui assurent, jusqu'à 6 mois durant, l'énergie nécessaire pour remplir sa mission.

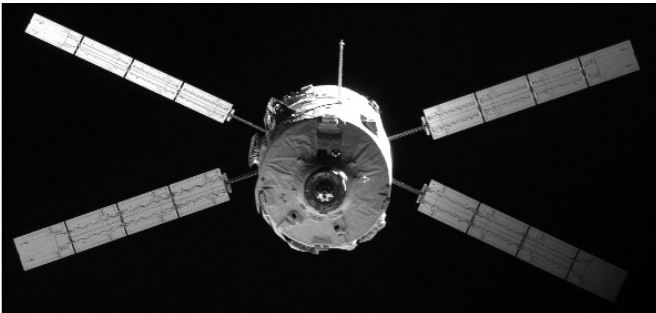


Figure 3 : L'ATV Jules Verne, lors de son rendez-vous avec l'ISS, le 3 avril 2008

L'architecture du système a tout d'abord été capturée en SysML avec l'outil Rhapsody d'IBM-SW. Les exigences textuelles ont été saisies dans des diagrammes d'exigences (REQ) puis des diagrammes de cas d'utilisation (UC) ont été définis de façon à capturer l'architecture fonctionnelle du système. Des diagrammes de séquences (SD) ont été utilisés pour capturer informellement le comportement dynamique du système. L'architecture statique a été classiquement capturée par des diagrammes de définition de blocs (BDD) et des diagrammes internes de blocs (IBD). Et enfin des diagrammes de machines à états (STM) décrivent précisément le comportement du véhicule.

Ce modèle système couvre à la fois la partie logiciel (gestion de mission, procédure de déploiement des panneaux solaires, gestion de la tolérance aux fautes) et la partie avionique, composée de plus d'une soixantaine d'équipements (4 unités de distribution de puissance, 2 convertisseurs numériques

analogiques, 2 électroniques de pilotage, 4 unités de contrôle thermique, 32 couteaux thermiques, 4 câbles physiques, 8 moteurs). Le comportement nominal de chaque équipement ainsi que le comportement en cas de défaillance ont été modélisés de façon à simuler le comportement du système en cas de panne et de vérifier l'objectif de tolérance à une faute.

Des preuves formelles sont en cours pour couvrir la soixantaine de cas de panne possibles du système.

L'architecture ainsi que les automates du logiciel de la gestion des panneaux solaires de l'ATV ont été modélisés en SCADE Suite puis vérifiés dans un premier temps par simulation et dans un second temps par preuve formelle. Ce modèle a été complété par une description en SPARK des algorithmes contrôlant la position des panneaux solaires (ceux-ci pouvant tourner autour de leur axe de façon à s'orienter vers le soleil et à optimiser l'énergie reçue).

Le code SPARK généré automatiquement depuis le modèle SCADE Suite a été intégré avec le code manuel, et les premières expérimentations sont en cours pour formellement prouver des propriétés de sûreté sur ce code de vol.

7. CONCLUSION

Les difficultés rencontrées lors du développement de systèmes complexes critiques sont bien connues : découverte tardive des erreurs d'architecture système, interfaces des composants mal définies, ambiguïtés dans les descriptions entraînant de mauvaises interprétations par les équipes de développement. Le processus de développement proposé par le projet FMDE intégré dans l'approche TASTE et assisté par des outils de vérification et de transformation de modèles permet d'éviter la plupart de ces écueils : formalisation de chaque étape du développement (permettant d'éviter toute ambiguïté et mauvaise interprétation), transformation de modèle et vérification au plus tôt par simulation et preuve formelle.

Nous sommes convaincus que ce processus et ces outils participeront à la résolution de la crise du logiciel, aussi bien dans le domaine spatial que dans d'autres domaines développant des systèmes complexes critiques.

8. RÉFÉRENCES

- [1] ASSERT project official website: www.assert-project.net
- [2] John Barnes : *High Integrity software: The SPARK approach to safety and security* ; Addison-Wesley. ISBN 0-321-13616-0, 25 avril 2003
- [3] Éric Conquet, Maxime Perrotin, Pierre Dis-
sieux, Thanassis Tsiodras et Jérôme Hugues : ►

- *The TASTE Toolset: turning human-designed heterogeneous systems into computer built homogeneous software* ; Embedded Real-Time Software and Systems (ERTSS'2010)
- [4] Emmanuel Hiron et Philippe Miramont : *Process-based on SysML for new launchers system and software developments* ; Data Systems In Aerospace (DASIA'2010)
- [5] IF website: <http://www-if.imag.fr/>
- [6] IFx-OMEGA website: <http://www.irit.fr/ifx>
- [7] David Lesens : *Use of the formal method SCADE for the specification of safety critical software for space application* : Data Systems In Aerospace (DASIA'2001)
- [8] Iulian Ober, Ileana Ober, Susanne Graf et David Lesens : *Projet OMEGA : Un profil UML et un outil pour la modélisation et la validation de systèmes temps réel* ; Génie Logiciel, n° 73, pp. 33-38, juin 2005
- [9] Iulian Ober et Iulia Dragomir : *OMEGA2: A new version of the profile and the tools* ; UML&AADL'2010 - 15th IEEE ICECCS, IEEE, pages 373-378, 2010
- [10] Iulian Ober et Iulia Dragomir : *Unambiguous UML composite structures: the OMEGA2 experience* ; SOFSEM 2011 - International Conference on Current Trends in Theory and Practice of Computer Science, Springer, LNCS 6543, pp. 418-430
- [11] SCADE Suite: <http://www.esterel-technologies.com/products/scade-suite/>
- [12] OMG Systems Modeling Language (OMG SysML™) version 1.1 (novembre 2008 - OMG Document Number: formal/2008-11-02)

175 mm