15-A-17

システム記述言語（AADL）による複合システム設計[1]

～航空機前方車輪の回転数から速度を

計測・記録・表示するシステムへの適用～

1. 適用した技術や手法の概要

1.1. 背景

航空機の前方にある車輪の回転数から航空機の地上走行時の速度を計測・記録・表示するシステムをシステム記述言語 AADL（Architecture Analysis & Design Language）[2]を利用し、開発した事例を紹介する。この事例は、2011 年に実施された「Theorem」（高信頼なソフトウェア開発のために必要な形式手法、ソフトウェア検証、数学の形式化および証明の計算機による検証に関する研究会）の第２回ワークショップで出された課題提案に基づいて開発されたシステムに記録機能と表示機能を加えたシステムとして開発したものである。

航空機などの複雑なシステムは、複数のコンポーネントで構成される。各コンポーネントは、複数の開発組織で異なる技術を使い、個別に開発・検証される場合が多い。したがって、個別に開発・検証されたコンポーネントを複合システムとして統合する際には、コンポーネントレベルの手直しや再検証が必要になる。

今回のシステムは小規模ではあるが、車輪回転数から速度演算を行うコンポーネント、速度を記録するコンポーネント、速度と記録を表示するコンポーネントなどの複数のコンポーネントで構成される。このような複数コンポーネントで構成される複合システムを、手直しや仕様の変更をすることなく開発する手法について述べる。

1.2. 適用技術

複数のコンポーネントを統合する場合、コンポーネントの属性の維持と管理は重要な課題となる。一般的にトップダウンとボトムアップのアプローチがあるが、複雑なシステムの設

---

[1] 事例提供: ISAE(Institut Supérieur de l'Aéronautique et de l'Espace) Space and Aeronautics Engineering　Jérôme Hugues 氏
「航空機産業の町、フランス南部のトゥールーズ。その代表格エアバスを人材供給・研究の面で支える航空宇宙高等学院(ISAE) ～」（2014 年 3 月 20 日付　日本経済新聞 電子版より引用）

[2] 米国 SAE が開発したアーキテクチャ言語 AADL：主にリアルタイムシステムのアーキテクチャを分析・設計するために使用されるモデリング言語。システムのアーキテクチャを詳細に記述することが容易。

計においては、トップダウンとボトムアップのアプローチが混在して使われる。ボトムアップのアプローチでは、既存のコンポーネントが再利用され最終システムに統合されるが、コンポーネント間相互の関係が決められていない場合には動作が保障されないことがある。

本事例では、システムの開発プロセスとしてトップダウンのアプローチを適用した。システムレベル（アーキテクチャレベル）の仕様の定義には AADL を使用した。AADL を採用することで、複数コンポーネントからなる複合システムをシステムレベルで記述することが可能になる。これにより、システムレベルの不具合（コンポーネント間のインターフェースの不一致、機能重複や機能欠落など）の確認ができる。

また、AADL を使用しシステムレベルの仕様を定義／確認した後、複数のコンポーネントに分解して詳細化を進める設計手法を採用した。これにより、複数のコンポーネントからなる複合システムの整合性を設計の早い段階で確認することができるほか、システムレベルで確認した仕様や属性がコンポーネントの詳細化の各工程でも維持、継承される。

さらに、複合システムのコンポーネントを開発する際に必要となる制約条件は Design by Contract[3] をサポートする言語（AADL の付属言語である REAL[4] や BLESS[5]、Simulink[6] のアサーション記述、SPARK[7] の契約記述）を利用し記述することとした。

設計の手順と適用した技術の概略を次に示す。

（1）　AADL を使用しシステムレベルの仕様を記述

システムの構造的な制約条件は、AADL の付属言語（BLESS や REAL のアサーション[8]）で記述する。

（2）　AADL モデルを分解し、Simulink モデルのコンポーネントを作成

構造的制約条件を、Simulink の制約条件（アサーション）に変換し、Simulink のモデルと制約条件でコンポーネントの機能と属性を検証する。

（3）　Simulink のモデルから SPARK のコードを自動生成

Simulink の制約条件から SPARK コードの制約条件(契約[9])を自動生成し､SPARK

---

[3] Design by Contract（契約プログラミング）は、形式言語により使用条件と機能要件を定義することでソフトウェアのコンポーネントを開発する手法[2]

[4] Requirement Enforcement Analysis Language アーキテクチャモデルの構造的制約条件を表現する

[5] Behavior Language for Embedded Systems with Software アーキテクチャモデルの構造的制約条件を表現する

[6] マルチドメインシミュレーションとモデルベース デザインのためのブロック線図環境

[7] 高信頼性ソフトウェアの開発に使用されるプログラミング言語。Ada 83 を元に Southampton 大学で英国防衛省の支援を受け開発され、その後 Program Validation 社、さらには Praxis HIS 社（現在の Altran 社）により拡張、改善された。

[8] プログラミング言語などの仕様・機能の１つで、プログラムの前提として満たされるべき条件を記述し、実行時にそれが満たされていない場合にエラーや例外を発生させたり、メッセージを表示して処理を中断したりする機能[3]

[9] ソースコードの中に記述されるプログラムが満たすべき仕様（契約）

のコードと制約条件で形式検証する。

1.3. 開発対象と使用言語・ツール

ここでは、開発対象と開発に使用した言語、ツールについて具体的に示す。

（1） 複合システム設計

速度演算コンポーネント、記録コンポーネント、表示コンポーネントからなる複合システムを AADL で記述した。また、各コンポーネントの制約条件を AADL に付属する REAL や BLESS 言語を使い記述した。これにより、トップダウンの開発プロセスを通じて制約条件を維持しながらステップバイステップで各コンポーネントの詳細化を進めることができた。

複合システムの AADL 記述から、AADL のオープンツールである Ocarina を使用し、Ada のソースコード（Ravenscar Profile[10]）を自動生成した。さらに、生成された Ada のソースコードに対し、静的アナライザツールである CodePeer[11] を使いマニュアルでのレビューを実施することで AADL 記述のエラーの可能性を検出した。

（2） 速度演算コンポーネント

複合システム設計で記述された速度演算機能の AADL 記述は、Simulink の速度演算コンポーネントに変換された。また、BLESS で記述された構造的制約条件は Simulink のアサーションに変換された。さらに、変換されたアサーションを利用し、コンポーネントの属性を検証した。

速度演算コンポーネントの Simulink モデルは、SPARK の認証済みのツールを使い SPARK コードに自動変換された。変換された速度演算コンポーネントは、SPARK により形式検証された。

（3） 記録コンポーネント

複合システム設計で記述された記録機能の AADL 記述は、SPARK のモデルに変換された。BLESS で記述された構造的制約条件は SPARK の制約条件（契約）にマニュアル変換された。SPARK の制約条件を使い、記録機能の SPARK モデルが形式検証された。

（4） 表示コンポーネント

複合システム設計で記述された GUI 機能の AADL モデルは、自然言語に変換されその後 Ada で実装された。実装された Ada のコードに対し、CodePeer を使用し、半自動でのピアレビューを実施した。

---

[10] 安全重視のリアルタイム システムを記述する Ada プログラミング言語のサブセット
http://blogs.windriver.com/parkinson/2006/10/the_real_ravens.html

[11] Ada のソースコードアナライザ http://www.adacore.com/codepeer

表 15-A-17-1 に使用言語・ツール一覧を示す。

表 15-A-17-1　使用言語・ツール一覧

開発対象	使用言語	言語の使用目的	使用ツール
複合システム設計	AADL	複合システムの仕様定義	Ocarina（Ravenscar を生成）
	Ravenscar	AADL の形式検証	CodePeer（Ravenscar の半自動レビュー）
	REAL、BLESS	コンポーネントの制約条件記述	
速度演算コンポーネント	Simulink	速度演算コンポーネントの機能確認（AADL 記述から変換）	Simulink（Simulink モデルのシミュレーション）
	Simulink	速度演算コンポーネントの制約条件記述（アサーション）	
	SPARK	速度演算コンポーネントの形式検証（Simulink モデルから変換）	SPARK（Simulink モデルから SPARK モデルの自動生成）
			SPARK（SPARK モデルの形式検証）
	SPARK	速度演算コンポーネントの制約条件記述（契約）	
記録コンポーネント	SPARK	記録コンポーネントの形式検証（AADL 記述から変換）	Simulink（Simulink モデルのシミュレーション）
表示コンポーネント	自然言語	表示コンポーネントの機能説明（AADL 記述から変換）	
	Ada	実装	CodePeer（Ada コードのピアレビュー）

2. 適用した技術や手法の導入理由

複合システムにおいては、各コンポーネントに最も適した開発方法が採用されるため、検証方法として様々な方法が使われている。したがって、開発・検証された各コンポーネントを１つのシステムに統合する際には、複合システムとして使用できないことになる。

航空・宇宙、自動車産業でよく使用されている AADL などのシステム記述言語を使用して複合システムを設計すれば、このような問題を解決することが可能となる。

3. 技術や手法を適用するための工夫

本事例で採用したトップダウンのアプローチでは、システムは複数のコンポーネントに分解されるとともに、各コンポーネントが個別に開発され、統合される（図 15-A-17-1）。

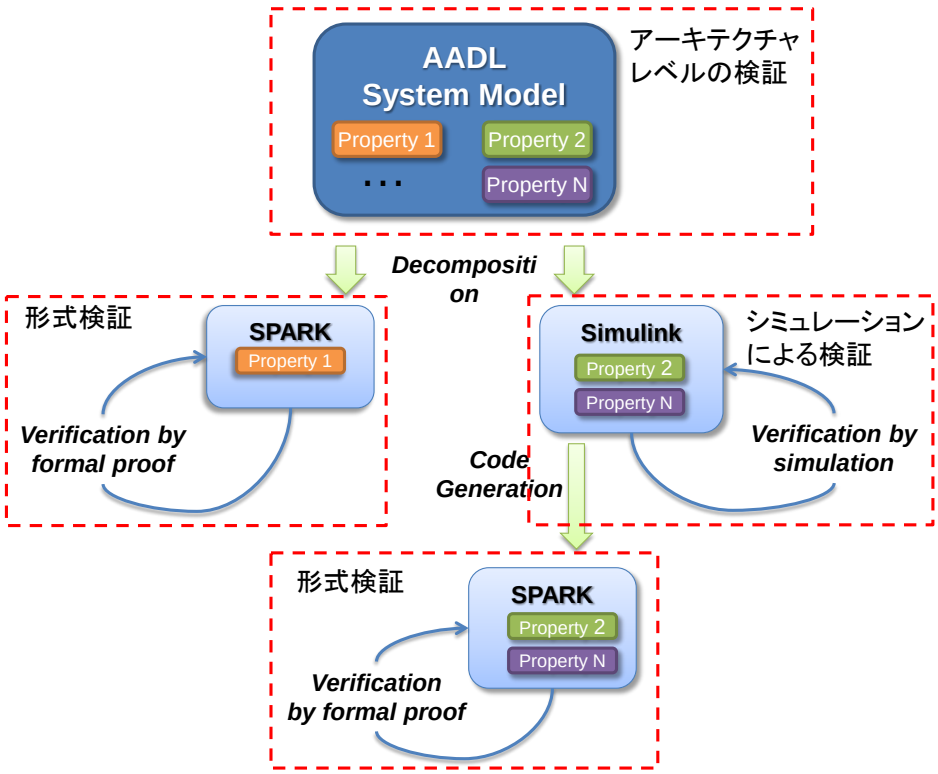
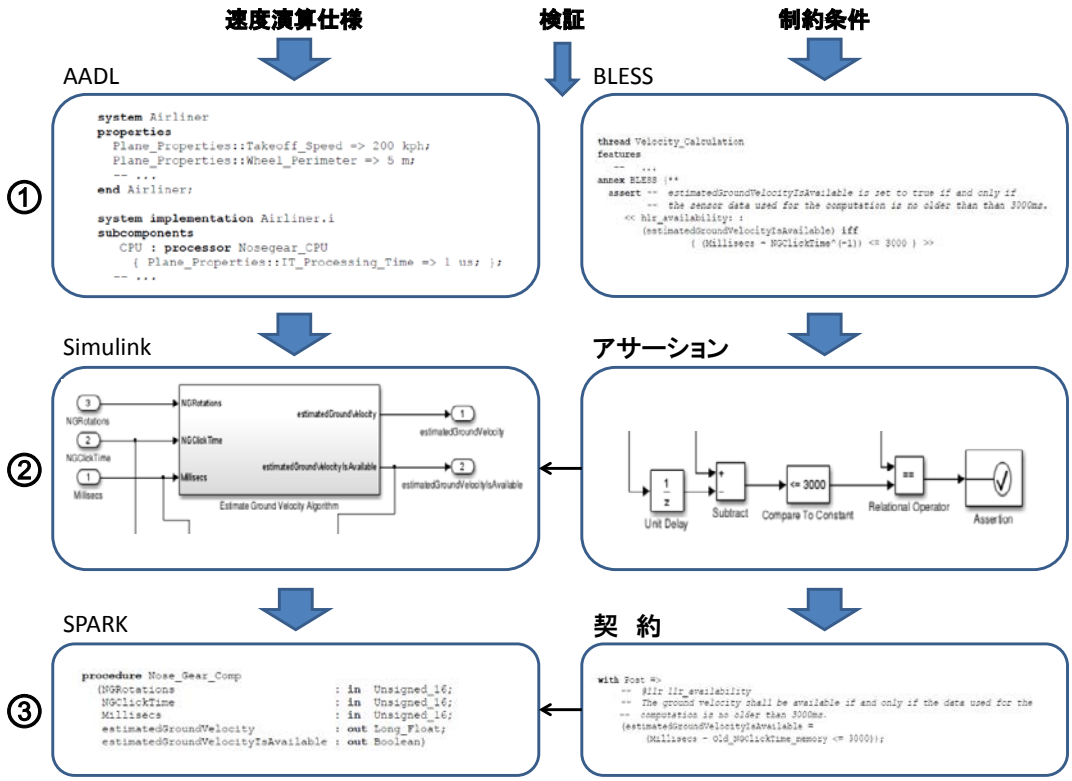


図 15-A-17-1　トップダウンのアプローチ

トップダウンのアプローチでは、属性の維持は難しく、仕様が正確に実装されない場合がある。例えば、Simulink のモデルに落とし込むとき、システム仕様で求められる性能を保証できないなどの問題が起こる可能性がある。そこで、このような問題を避ける工夫として、AADL の付属言語である REAL や BLESS を使い、詳細化に必要な制約条件を記述することとした。その結果、上流工程で記述した制約条件を使い下流工程で検証できるようになった。

仕様と制約条件から演算モデルを開発する手順について、速度演算の例（BLESS を使用）を図 15-A-17-2 に示す。



① 速度演算の仕様を AADL で記述し、制約条件は BLESS で記述

② AADL の速度演算仕様は Simulink の速度演算モデルに変換、BLESS の制約条件は Simulink のアサーションに変換。Simulink の速度演算モデルをアサーションの制約条件で検証。

③ Simulink の速度演算モデルは SPARK の速度演算モデルに変換、Simulink のアサーションは SPARK の契約に変換。SPARK の速度演算モデルを SPARK の契約で検証。

図 15-A-17-2　速度演算モデルの開発手順

これにより、BLESS で記述した制約条件を維持しながら、AADL で記述した速度演算仕様を Simulink および SPARK のモデルに変換することができた。

4. 技術や手法の適用効果

規模は小さいが実際のシステムに利用できる実践的な開発について、AADL を利用した開発プロセスの適用を実証した。これにより、トップダウンアプローチの技法において、以下のことを確認した。

- AADL から Simulink モデル、Simulink モデルから SPARK コード、AADL から SPARK コードに変換するプロセス（自動化が困難なものについては、マニュアルで実施）
- アーキテクチャレベルで検証されたコンポーネントの仕様や属性を、異なる技術やプ

ロセスを使用する詳細化の各工程で維持できること

したがって、上記手法を採用すればシステムレベルで検証することができ、初期工程での不具合の特定と解決が可能となる。

5. 今後の課題

トップダウンのアプローチでは、実装コードの自動生成は比較的簡単にできるが、属性や制約条件の変換はより複雑な作業になる。現状では、Simulink から SPARK への属性の変換は部分的に可能であるが、システムの構造的制約条件を記述する AADL の BLESS や REAL を Simulink や SPARK の制約条件に自動変換することはできない。AADL の BLESS は比較的新しい言語のために、Simulink や SPARK に変換するツールは現時点ではない。さらに、BLESS の文法には、時間の概念が含まれているので、自動変換は困難である。

言語間での属性変換の自動化は、複数の抽象レイヤーで属性を維持するという完全自動化のプロセスに向けた重要な要素と考えられる。今後は、種々のプロセスにおける自動化の比率を高めることが課題である。

6. 考察

6.1. 品質について

今回の開発では品質の改善を示す定量的な評価は行っていない。このため、品質の改善が実現した根拠となるデータや情報はない。一般的に AADL を用いたモデルベース開発においては、設計の初期段階で不具合を検出することができるため、品質を改善することができる。

6.2. 生産性について

AADL では各コンポーネントの詳細設計以前にシステムレベルで検証が可能なため、通常の検証コストを削減することができる。

図 15-A-17-3 に米国立標準技術研究所（NIST：National Institute of Standards and Technology）による分析の例を示す。この分析によると、プロジェクトのテスト段階で不具合を発見し解決するコストは、設計段階で不具合を発見し解決するコストの 5 倍から 30 倍になるとされている。

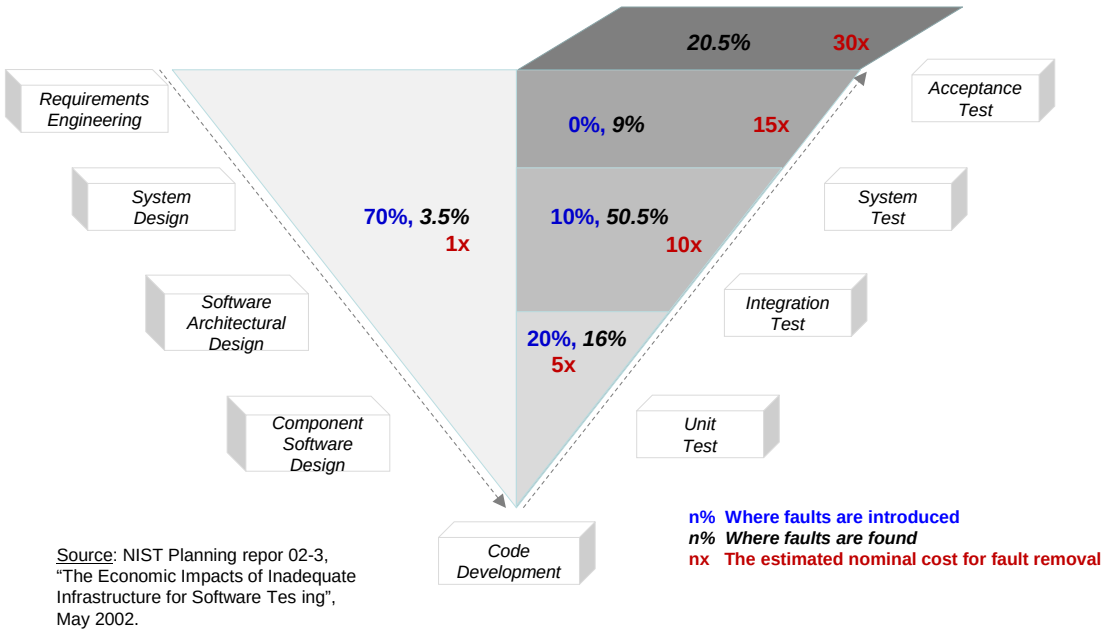


図 15-A-17-3　上流工程での不具合の発見と解決の効果[4][5]

Jerome Hugues 氏によれば、具体的な効果の測定はしていないが、一般的に、設計初期での不具合の発見により、下流工程での工数が大幅に減少するとのことであった。

また、実装コードや属性の自動変換、制約条件（アサーションや契約）の自動変換、検証のプロセスの自動化がされることで、生産性はさらに改善すると考えられる。

AADL などのモデル開発言語においては、構成システムやコンポーネントの契約や制約条件を明確に定義し、記述する必要がある。このためにかかる時間は、生産性の面ではマイナスになるものの、その後の工程で検証の自動化などが図れるため、全体として生産性は向上すると考えられる。

6.3. 検証への影響

システムのアーキテクチャは、AADL により記述されシステムレベルでの検証がなされた（Ada の Ravenscar Profile に変換し検証）。

システムレベルのコンポーネントの属性は、各コンポーネントに割り当てられ、コンポーネントレベルで検証された（Simulink のシミュレーションや SPARK による形式検証）。属性や制約条件を維持しながら、コンポーネント レベルの検証を実施し、コンポーネントが統合された場合でも、コンポーネントでの検証結果が有効であることが確認できた。

6.4. 水平展開状況

AADL は、航空・宇宙、自動車産業で利用されているものの十分に普及しているとはいえない。さらに、上記産業に加え、一般的な手法として普及するためには、属性や制約条件をサポートするツール、およびコンポーネント間の変換を自動化するツールなどの開発が重要である。

7. まとめ

開発の現場では、複数の技術や手法、ツールが使われている。開発プロセスが複数の組織や企業にまたがる場合はなおさらである。さらには、開発の詳細レベルも、プロセスや組織によってさまざまである。開発システムの規模が大きく、複雑になるにつれ、コンポーネント数や、利害関係者が増加する。このような環境では、開発組織間、コンポーネント間のすり合わせの重要性が高まっている。

AADL を用いた設計することにより、複合システムの設計・検証においても、複数の組織・プロセスで開発された複数のコンポーネントの属性を維持できることが分かった。

参考文献

[1] System to Software Integrity: A Case Study
http://www.spark-2014.org/uploads/Nosegearpaper_2.pdf
[2] Adapting the PSP to Incorporate Verified Design by Contract, Software Engineering Institute, http://www.sei.cmu.edu/podcasts/podcast_episode.cfm?episodeid=294150
[3] IT 用語辞典 e-Words http://e-words.jp/w/E382A2E382B5E383BCE382B7E383A7E383B3.html
[4] NIST Planning Report 02-3, The Economic Impacts of Inadequate Infrastructure for Software Testing, May 2002
[5] DISC/IpSC (Département d'Ingénierie des Systèmes Complexes /Ingénierie pour les systèmes critiques) : Jérôme Hugues, Institut Supérieur de l'Aéronautique et de l'Espace, 2015

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（IPA/SEC）