Верификация и валидация при разработке в соответствии с IEC 61508

Мирко Конрад
The MathWorks, Inc., Натик, Массачусетс, США

Гидо Сэндманн
The MathWorks GmbH, Мюнхен, Германия

Перевод Михаил Песельник
Департамент MathWorks, Россия

РЕЗЮМЕ

Модельно-ориентированное проектирование адресует проблемы, связанные со сложностью программного обеспечения и производительностью труда. По этой причине модельно-ориентированное проектирование и генерация кода производственного качества интенсивно используются в автомобильной индустрии. В последнее перед инженерами встает вопрос соответствия таким стандартам, как IEC 61508 – и модельно-ориентированное проектирование помогает решать эту задачу.

Для систем внутри автомобиля обычно применяется стандарт IEC 61508-3. Чтобы показать соответствие стандарту, соответствующие цели и рекомендации, обозначенные в IEC 61508-3, должны быть привязаны к процессам и инструментам модельно-ориентированного проектирования.

В этой статье обсуждаются процессы верификации и валидации при разработке автомобильных программных компонентов, которые должны соответствовать требованиям IEC 61508 с использованием модельно-ориентированного проектирования.

Прим. переводчика: Несмотря на то, что с 2011 года в автомобильной индустрии принят стандарт ISO 26262, данная статья всё еще является актуальной. Причем как с точки зрения стандарта IEC 61508-3 (который применяется для разработки программных электронных систем в самых разных индустриях), так и с точки зрения ISO 26262, который является производным от IEC 61508.

ВВЕДЕНИЕ

В последние десятилетия сложность программного обеспечения автомобилей неизменно увеличивается. Количество функций, которые выполняются каждым блоком управления, и связи между блоками, существенно увеличились.

Процесс разработки играет важную роль для решения этих проблем. Модельно-ориентированное проектирование для разработки автомобильных блоков управления наряду с V-моделью получают широкое применение, поскольку предлагают несколько преимуществ. Моделирование улучшает взаимодействие между производителями системы и поставщиками, а также между инженерами в их проектах.

В центре модельно-ориентированного проектирования находится исполняемая модель, представляющая разрабатываемый программный встраиваемый компонент. Эта модель служит в качестве основного представления в нескольких фазах процесса разработки. Первоначальная исполняемая модель (исполняемая спецификация) уточняется и детализируется до тех пор, пока не становится детальным планом (blueprint) для окончательной реализации. В дополнение, исполняемые модели могут быть использованы для различных мероприятий верификации и валидации.

Модельно-ориентированное проектирование адресует проблемы, связанные со сложностью программного обеспечения и производительностью труда. По этой причине модельно-ориентированное проектирование интенсивно используется при разработке программного обеспечения. Все большее число проектов должны соответствовать стандартам, поскольку современные блоки управления и их прикладное ПО напрямую взаимодействуют с такими системами, как тормозная и рулевая.

Производители и поставщики недавно стали применять модельно-ориентированное проектирование для разработки встраиваемого ПО приложений, которые должны соответствовать стандарту IEC 61508. В качестве примеров можно привести программные компоненты электромеханической рулевой системы [JSB+08] для Volkswagen Tiguan [FMC08].

Для автомобильных систем, IEC 61508-3 часто рассматривается как высочайший технический уровень системы. Для демонстрации соответствия стандарту, цели и рекомендации, обозначенные в IEC 61508-3, должны быть привязаны к процессам и инструментам модельно-ориентированного проектирования.

В продолжение этой статьи обсуждается процесс верификации и валидации для приложений, которые должны соответствовать IEC 61508 с использованием модельно-ориентированного проектирования.

ПРОЦЕСС ВЕРИФИКАЦИИ И ВАЛИДАЦИИ МОДЕЛЕЙ И СГЕНЕРИРОВАННОГО КОДА

Как и другие стандарты, IEC 61508-3 описывает **верификацию и валидацию применительно к конкретному приложению** вне зависимости от используемых инструментов и парадигмы разработки.

При верификации и валидации конкретных приложений, реализованных при помощи модельно-ориентированного проектирования, возникают следующие два вопроса:

1. Корректно ли реализует модель свои (текстовые) требования?
2. Корректно ли реализует объектный код, работающий в блоке управления, поведение модели?

Для того чтобы упростить использование модельно-ориентированного проектирования в контексте IEC 61508, был разработан шаблон рабочего процесса, описывающий мероприятия верификации и валидации, необходимые для соответствия IEC 61508-3.

Следуя приведенным выше вопросам, для приложений, разработанных с использованием модельно-ориентированного проектирования и генерации кода, рабочий процесс будет разбит на следующие два шага (см. [Ald01]):

1. Демонстрация того, что модель является корректной и удовлетворяет всем требованиям.
2. Демонстрация того, что сгенерированный код является эквивалентным модели.

Первый шаг это **верификация проекта ПО**, который сочетает соответствующие техники верификации и валидации на уровне модели. Второй шаг это **верификация кода**.

ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ НА УРОВНЕ МОДЕЛИ (ВЕРИФИКАЦИЯ ПРОЕКТА ПО) – Цель верификации проекта ПО заключается в том, чтобы

обеспечить уверенность в модели, которая затем используется для генерации производственного кода. Этот шаг осуществляется на уровне модели, т.е. до генерации кода.

Следуя букве IEC 61508-3, часть рабочего процесса, касающаяся верификации проекта ПО, состоит из комбинации рассмотрения (инспекции), статического анализа и тщательного функционального тестирования на уровне модели [Con08]. Совместно, эти мероприятия помогают обеспечить уверенность в том, что проект ПО удовлетворяет связанным требованиям. Результатом этого процесса является **золотая модель**, т.е. достаточно верифицированная и валидированная модель, которая реализует требования и не содержит непредусмотренной функциональности.

Требования для верификации проекта ПО могут быть найдены в IEC 61508-3 п. 7.4.6 (реализация исходного кода), 7.4.7 (тестирование программных модулей) и 7.4.8 (тестирование интеграции ПО).

Рассмотрение и статический анализ на уровне модели – Подсистемы модели, являющиеся модулями (компонентами модели), должны быть рассмотрены. Если это выполнимо, то ручные **рассмотрения модели** должны быть поддержаны автоматическим **статическим анализом** модели.

Должны использоваться **руководства по моделированию** и должна быть проведена оценка соответствия этим руководствам. Конструкции моделирования, которые не подходят или не рекомендуются для генерации производственного кода, не должны использоваться.

Поддержка со стороны инструментов: Рассмотрения моделей облегчаются с использованием отчетов или веб-представлений, сгенерированных при помощи Simulink Report Generator. Соответствие руководствам по моделированию может быть частично установлено с использованием предопределенных или собственных проверок на стандарты моделирования в инструменте Model Advisor [Beg07].

Тестирование модулей и интеграции на уровне модели – Компоненты модели должны быть функционально протестированы с использованием систематически получаемых тестовых векторов. Целью **модульного тестирования** является демонстрация того, что каждый компонент модели осуществляет свои заданные функции и не осуществляет никаких непредусмотренных функций.

После завершения модульного тестирования, должно быть осуществлено **тестирование интеграции** модулей с заданными тестовыми векторами, т.е. стадии интеграции модели должны быть протестированы в соответствии с заданными интеграционными тестами. Эти тесты должны показать, что все модули модели и подсистемы модели взаимодействуют корректно для осуществления заданных функций и не осуществляют никаких непредусмотренных функций.

Поддержка со стороны инструментов: Simulink Verification and Validation поддерживает различные аспекты тестирования модели.

ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ НА УРОВНЕ КОДА (ВЕРИФИКАЦИЯ КОДА) - Используется **валидация трансляции путем систематического тестирования (тестирование трансляции)** для демонстрации того, что семантика выполнения модели сохраняется во время генерации кода, компиляции и компоновки.

Выражаясь техническим языком, применяется **тестирование численной эквивалентности** между моделью, используемой для генерации производственного кода и исполняемого файла, полученного из сгенерированного кода, а также дополнительные меры для демонстрации **отсутствия непредусмотренной функциональности**.

На рисунке 1 показан обзор предлагаемого процесса валидации трансляции сгенерированного кода.

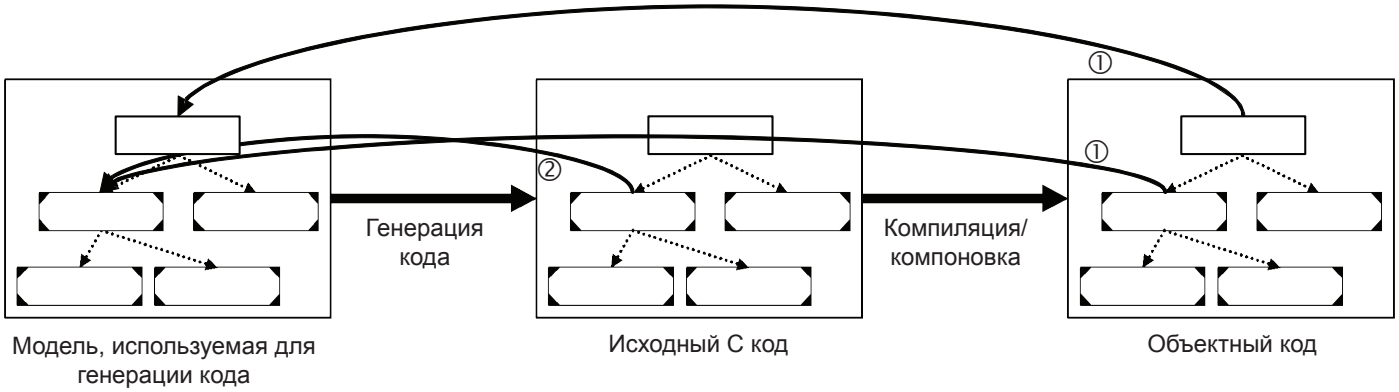

Рис. 1: Процесс валидации трансляции.

① *Тестирование эквивалентности*
② *Предотвращение непредусмотренной функциональности*
a) Сравнение покрытия тестами модели и кода
b) Анализ трассируемости

Тестирование численной эквивалентности – Тестирование эквивалентности осуществляется для демонстрации численной эквивалентности между моделью и сгенерированным кодом. Тестирование эквивалентности между моделью и результирующим объектным кодом (также известное как сравнительное или back-to-back тестирование) представляет основную часть процесса верификации кода.

Тестирование эквивалентности осуществляется путем стимуляции как модели, так и объектного кода, полученного из модели, одинаковыми тестовыми векторами. Валидность процесса трансляции, т.е. того, была ли сохранена семантика модели в процессе генерации кода, компиляции и компоновки, определяется путем сравнения реакций модели и сгенерированного кода на идентичные тестовые вектора *i(t)*.

Точнее говоря, результаты симуляции модели, используемой для генерации производственного кода $o_{SIM}(t)$, сравниваются с результатами выполнения скомпилированного кода $o_{CODE}(t)$.

На рисунке 2 показан предлагаемый подход к валидации процесса трансляции. Более детальное описание процедур тестирования эквивалентности может быть найдено в [SC03, SC05, SCD+07].

Тестирование численной эквивалентности является уникальным в том смысле, что не нужно предоставлять ожидаемые выходы для тестовых векторов [Ald01]. Таким образом, тестирование эквивалентности хорошо поддается автоматизации.

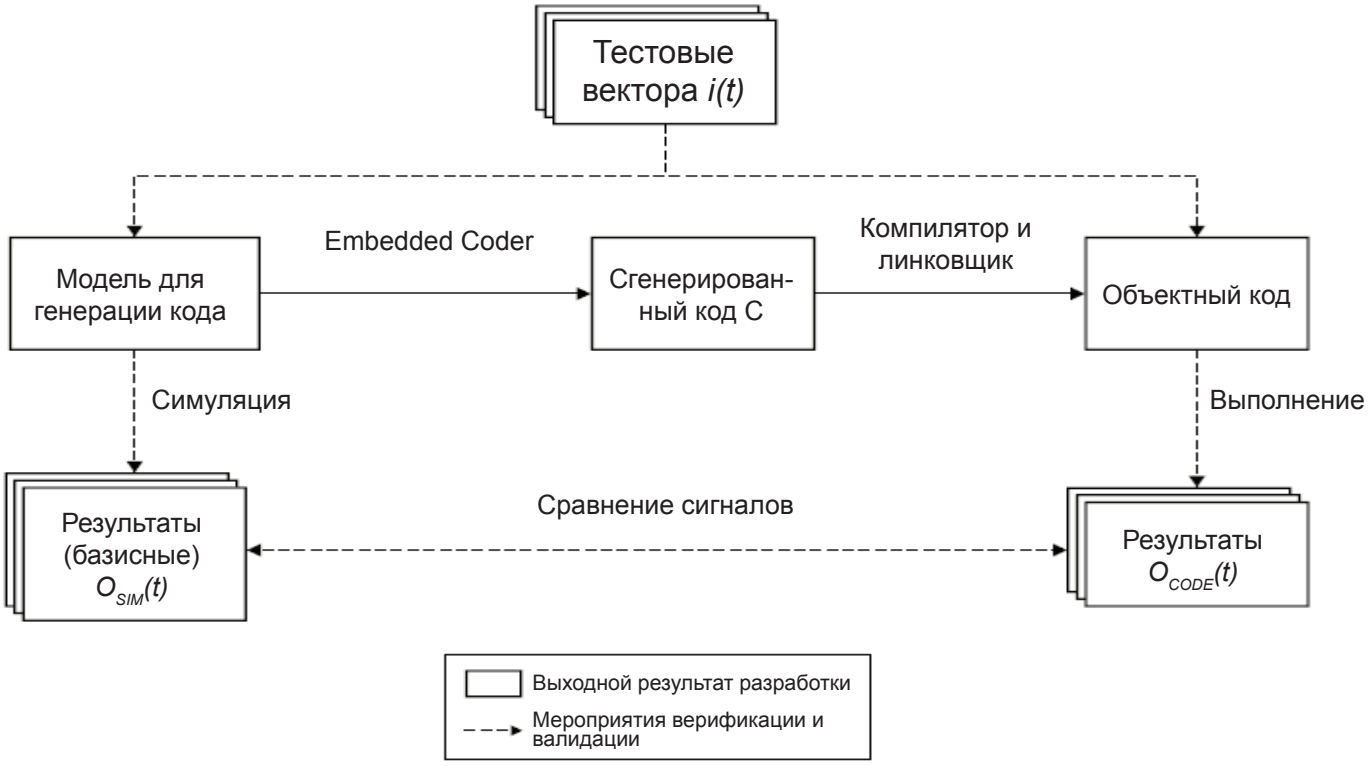

Рис. 2: Тестирование численной эквивалентности.

Следующие секции более детально описывают процедуры тестирования эквивалентности.

Генерация тестовых векторов для проверки эквивалентности: Валидная трансляция требует, чтобы выполнение объектного кода давало бы те же самые наблюдаемые результаты, что и симуляция модели для заданного набора тестовых векторов. Поскольку полное тестирование невозможно по причинам сложности, тестовые вектора должны быть достаточными для покрытия различных структурных частей модели.

Чтобы оценить степень покрытия модели, требуется установить определенные **метрики покрытия тестами** для уровней SIL 2 и выше [SS05]. Глубина и степень структурного покрытия модели должны быть увеличены для более высоких уровней SIL.

Поддержка со стороны инструментов: Анализ покрытия модели осуществляется при помощи Model Coverage Tool из Simulink Verification and Validation [Simulink Verification and Validation].

Тестовые вектора, получаемые на основании требований на уровне модели, могут быть повторно использованы для тестирования на эквивалентность. На рисунке 3 показано тестирование на эквивалентность индивидуальных и интегрированных модулей. $i_{COMP}(t)$ и $o_{COMP}(t)$ соответственно означают тестовые вектора и результаты для модели; $i_{INT}(t)$ и $o_{INT}(t)$ соответственно означают тестовые вектора и результаты для стадии интеграции.

Если покрытие, достигнутое при помощи существующих тестовых векторов, не является достаточным, то должны быть созданы дополнительные тестовые вектора.

Если для выбранных метрик не может быть достигнуто полное покрытие, то непокрытые части должны быть проанализированы с предоставлением обоснования. На практике, набор векторов может быть итеративно расширен с использованием анализа покрытия модели, пока не будет достигнут требуемый уровень покрытия.

Поддержка со стороны инструментов: Simulink Design Verifier используется для создания дополнительных тестовых векторов для тестирования эквивалентности [Simulink Design Verifier].

Выполнение тестов на эквивалентность: Тесты должны быть запущены как над моделью, из которой генерируется код, так и над объектным кодом, полученным из сгенерированного кода. Результирующий объектный код должен тестироваться в среде, которая максимально соответствует целевому окружению, в котором будет разворачиваться код.

Результирующий объектный код может исполняться либо на целевом процессоре, или на похожем процессоре – например, при помощи симуляции процессор-в-контуре (processor-in-the-loop, PIL), или на симуляторе целевого процессора (instruction set simulator, ISS). Если это возможно, то верификация в режиме PIL является предпочтительной.

Если исполнение результирующего объектного кода не осуществляется в целевом окружении, то различия между средой тестирования и целевым окружением должны быть проанализированы, чтобы убедиться, что они не влияют отрицательно на результаты.

Сравнение сигналов: После выполнения тестов, результаты симуляции модели $o_{SIM}(t)$ должны сравниваться с результатами выполнения сгенерированного кода $o_{CODE}(t)$. Результаты симуляции модели используются как эталонные. Они сравниваются с результатами, полученными при выполнении объектного кода.

Даже если трансляция модели Simulink или Stateflow в С код осуществлена корректно, не всегда можно ожидать идентичного поведения (эквивалентности). Возможные причины заключаются в ограниченной точности чисел в плавающей точке, эффектов квантования при использовании арифметики с фиксированной точкой и различиями между компиляторами. По этим причинам определение корректности должно основываться на допустимом эквивалентном поведении (допуске).

Необходимо выбрать подходящий алгоритм сравнения сигналов, который бы учитывал погрешности между результатами симуляции $o_{SIM}(t)$ и работы кода $o_{CODE}(t)$. Существует большое число алгоритмов сравнения, как простых - таких, как абсолютная разница, так и более сложных – таких, как метод разностных матриц.

Два вектора с результатами тестирования являются одинаковыми, если их разница с учетом выбранного алгоритма сравнения меньше или равна заданному порогу. Выбор алгоритма сравнения и порога зависит от приложения и должен быть задокументирован.

Simulink Fixed Point позволяет осуществлять симуляцию с точностью до бита для моделей в арифметике с фиксированной точкой и наблюдать эффекты ограниченной точности и диапазона для моделей Simulink и Stateflow [Simulink Fixed Point]. При использовании Real Time Workshop Embedded Coder, Simulink Fixed Point позволяет генерировать чистый целочисленный код из таких моделей. Сгенерированный код с точностью до бита совпадает с моделью, и обеспечивает точно такое же выполнение, как во время симуляции.

Одинаковость служит базой для определения функциональной эквивалентности, что определяет численную корректность трансляции модели в код: модель и код, сгенерированный из нее, считаются функционально эквивалентными, если симуляция модели и выполнение объектного кода, полученного из сгенерированного кода, привело к достаточно одинаковым результатам при использовании идентичных тестовых векторов.

Поддержка со стороны инструментов: Алгоритмы сравнения могут быть реализованы при помощи языка программирования, такого, как MATLAB. В [MEval] представлены различные алгоритмы для сравнения результатов тестов.

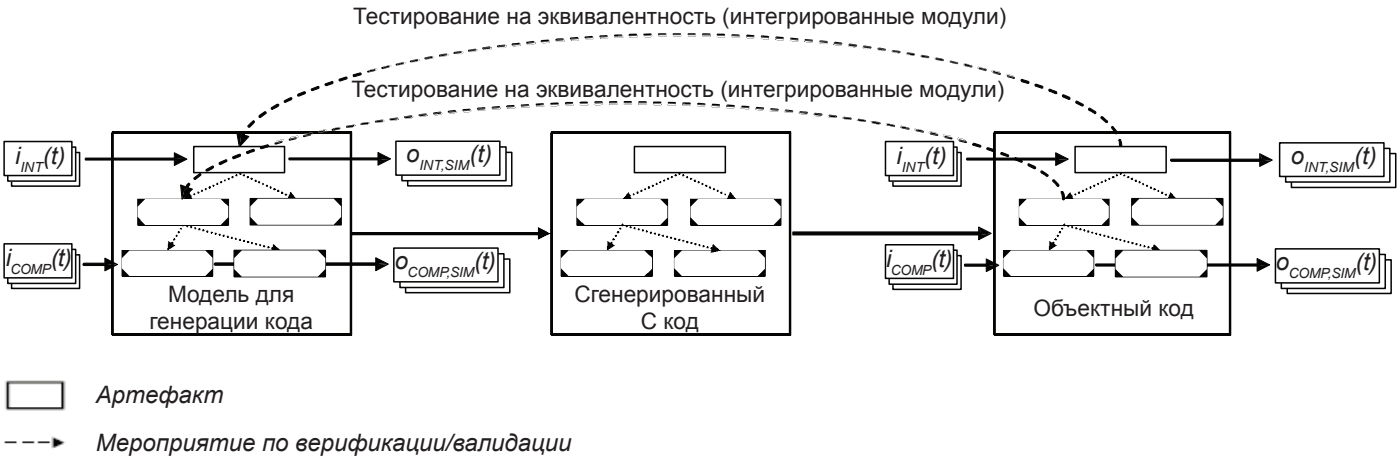

Рис. 3: Тестирование на эквивалентность индивидуальных и интегрированных модулей.

Предотвращение нежелаемой функциональности – Второе мероприятие в процессе верификации кода для демонстрации того, что сгенерированный C код не осуществляет никаких **непредусмотренных функций**.

Для достижения этой цели используются различные техники. Их целью является демонстрация структурной эквивалентности между моделью и сгенерированным исходным кодом.

Можно использовать как минимум одну из следующих мер для демонстрации того, что сгенерированный C код не осуществляет никаких **непредусмотренных функций**:

- Сравнение покрытия модели и кода
- Анализ трассируемости

Сравнение покрытия **модели и кода**: Если используется сравнение покрытия модели и кода, то измеряются степени покрытия модели и кода во время тестирования на эквивалентность, а затем сравниваются. Должна быть проведена оценка различий в сравниваемых метриках.

Сравниваемые метрики структурного покрытия должны анализироваться на уровне модели и кода. В соответствии с [BCS+03], покрытие решений на уровне модели и покрытие решений на уровне кода (decision coverage) должны анализироваться совместно.

Различия между покрытием модели и кода по соответствующим метрикам должны быть оценены. Если покрытие кода меньше, чем покрытие модели, то это может говорить о внесении непредусмотренных функций.

Поддержка со стороны инструментов: Информация о покрытии кода может быть получена из среды разработки (IDE) или применением независимых инструментов анализа покрытия кода.

Анализ трассируемости: **Анализ трассируемости** сгенерированного исходного C кода может быть осуществлен для обеспечения того, что все части кода трассируются обратно к модели, используемой для генерации производственного кода. В таком случае сгенерированный код является предметом **ограниченного рассмотрения**, которое фокусируется исключительно на аспектах трассируемости.

Нетрассируемый код должен быть оценен.

Поддержка со стороны инструментов: Автоматически сгенерированные комментарии для блоков Simulink могут использоваться для трассируемости сгенерированного кода [Real-Time Workshop Embedded Coder]. Отчет о трассируемости из Real-Time Workshop Embedded Coder устанавливает соответствие между блоками модели и сгенерированным кодом. В нем отображаются списки удаленных или виртуальных блоков, а также трассируемые блоки. Подсветка "код-к-блоку" генерирует гиперссылки в отображаемом исходном коде для быстрого просмотра блоков или подсистем, из которых был сгенерирован этот код. Подсветка "блок-к-коду" позволяет увидеть, как реализован в сгенерированном коде любой блок модели.

ЗАКЛЮЧЕНИЕ

Сейчас IEC 61508-3 является широко используемым стандартом для разработки встраиваемого ПО. Он определяет требования и ограничения для процессов разработки ПО и оценки качества. Эти требования применяются как к модельно-ориентированному проектированию, так и к традиционному процессу разработки. Однако реализация этих требований при использовании модельно-ориентированного проектирования требует особого подхода и создает специфические вызовы.

В этой статье обсуждаются процессы модельно-ориентированного проектирования и инструменты, адресующие цели IEC 61508-3.

Предложенный рабочий процесс для верификации и валидации моделей и сгенерированного кода может рассматриваться как конкретизация требований по верификации и валидации, описанных в IEC 61508-3, и использует преимущества автоматической генерации кода как неотъемлемой части модельно-ориентированного проектирования.

СПИСОК ЛИТЕРАТУРЫ

1. [Ald01] Aldrich, W., Coverage Analysis for Model- Based Design Tools, TCS 2001.
2. [BCS+03] Baresel, A., Conrad, M.,Sadeghipour, S., Wegener, J., The Interplay Between Model Coverage and Code Coverage, 11. European Int. Conf. on Software Testing, Analysis and Review (EuroSTAR '03), Amsterdam, Netherlands, 2003.
3. [Beg07] Begic, G., Checking Modeling Standards Implementation, The MathWorks News & Notes, June 2007.
4. [Bur04] Burnard, A., Verifying and Validating Automatically Generated Code, Int. Automotive Conference (IAC '04) Stuttgart, Germany, 2004, pp. 71-78.
5. [Con07] Conrad, M., Using Simulink and Real-Time Workshop Embedded Coder for Safety-Critical Automotive Applications, Proc. Workshop Modellbasierte Entwicklung Eingebetteter Systeme III (MBEES'07), Schloß Dagstuhl, Germany, 2007, pp. 41-50.
6. [Con08] Conrad, M., Model-Based Design for IIEC 61508: Towards Translation Validation of Generated Code, Proc. Workshop Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis, colocated with Software Engineering 2008, Munich, February 2008.
7. [EC07] Erkkinen, T. and Conrad, M., Safety-Critical Software Development Using Automatic Production Code Generation, Proc. SAE World Congress 2007, Detroit, USA, 2007.
8. [ECCert] www.mathworks.com/company/pressroom/articles/a rticle18304.html
9. Ekkehard Pofahl, Torsten Sauer, Oliver Busa.
10. [ECAVS] www.mathworks.com/company/pressroom/articles/a rticle17790.html
11. [Edw99] Edwards, P.D., The Use of Automatic Code Generation Tools in the Development of Safety- Related Embedded Systems, Proc. Vehicle Electronic Systems, ERA Report 99-0484, 1999.
12. [Embedded MATLAB] Embedded MATLAB feature page: www.mathworks.com/products/featured/embedded matlab
13. [FMC08] Fey, I., Müller, J., Conrad, M., Model- Based Design for Safety-Related Applications, Proc. Convergence 2008, Detroit, MI, USA, Oct. 2008.
14. [HC06] Harmon, R., Hote, C., Automatic Engine Control Code Generation with Integrated Automatic Static Code Verification, International Automotive Conference (IAC '04), Stuttgart, Germany, 2006.
15. [IEC61508-3] IEC 61508-3:1998, Int. Standard Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems - Part 3: Software Requirements, First edition, 1998.
16. [JSB+08] Thorsten Jablonski, Heiko Schumann, Carsten Busse, Heiko Haussmann, Udo Hallmann, Dirk Dreyer, Frank Schöttler, Die neue elektromechanische Lenkung APA-BS, ATZelektronik 01/2008 Vol. 3 (2008) 01, pp. 30-35.
17. [Model-Based Design] Model-Based Design Web page: www.mathworks.com/applications/controldesign/description
18. [MEval] MEval product page: www.itpower.de/meval_e.html