Инструментальные средства проектирования систем интегрированной модульной авионики

Буздалов Д.В., Зеленов С.В., Корныхин Е.В., Петренко А.К., Страх А.В., Угненко А.А., Хорошилов А.В.
{buzdalov, zelenov, kornevgen, petrenko, strakh, ugnenko, khoroshilov} @ispras.ru

Аннотация. Масштабы современных комплексов бортового авиационного оборудования таковы, что их проектирование становится невозможным без привлечения средств автоматизации. В настоящее время в мире в этой области имеются с одной стороны закрытые разработки крупных авиакомпаний, таких как Boeing и Airbus, а с другой стороны ряд открытых международных проектов с разной степенью зрелости, доступности исходного кода и документации. В настоящей статье представлена отечественная разработка открытой системы поддержки проектирования и верификации комплексов бортового авиационного оборудования осуществляемая в ИСП РАН совместно с ГосНИИАС в рамках государственной программы по развитию Интегрированной Модульной Авионики.

Ключевые слова: интегрированная модульная авионика; ИМА; моделирование систем; AADL.

1. Введение

Разработка современных систем авионики и других критических по безопасности систем управления требует развитой методической и инструментальной поддержки. За рубежом имеются соответствующие инструменты, но развитие таких высокотехнологичных отечественных отраслей, как авиастроение, не может опираться только на них в силу, по крайней мере, двух причин. Во-первых, такие инструментальные средства достаточно дороги, во-вторых, что, вероятно, более важно, они «закрыты» для развития и адаптации силами отечественных исследователей и инженеров, что ведет к еще большему отставанию наших технологий в данной области.

Средства проектирования, разработки, верификации и валидации систем типа авионики традиционно поддерживают подход разработки систем на основе моделей (Model Driven Engineering – MDE и Model Driven System Engineering - MDSE), поскольку методы моделирования в разных его видах: натурного, полунатурного, математического, - всегда культивировались в авиастроении и смежных отраслях [1]. В последние 20-30 лет в области разработки

программного обеспечения появился новый вид моделирования, связанный с исследованиями по формальным спецификациям программ и использованием так называемых формальных методов для анализа, в частности, для верификации программных систем. Системы авионики сейчас – это комплексы программно-аппаратных средств, поэтому методы и подходы, наработанные в области проектирования и анализа авионики и программных систем, естественно, должны обогащать друг друга. По этой причине опыт ИСП РАН в области использования формальных методов верификации сложных программных и аппаратных систем, таких как операционные системы и микропроцессоры, позволил достаточно быстро освоить еще одно направление – разработку средств проектирования и интеграции систем авионики, поскольку многие задачи в этой новой области могут быть решены на основе технологий моделирования и верификации, созданных в ИСП РАН ранее. Решающее значение в достижении полученных в этом направлении результатов оказала поддержка академика Е. А. Федосова, Г. А. Чуянова, И. В. Ковернинского и других сотрудников ГосНИИАС, работающих в рамках государственной программы по развитию Интегрированной Модульной Авионики (ИМА). В настоящей статье рассматривается опыт развития методов моделирования, синтеза и верификации сложных систем применительно к ИМА воздушных судов, но сфера потенциального применения технологии существенно шире.

Последующие разделы статьи построены следующим образом. В разделе 2 рассматриваются основные идеи подхода ИМА, новые задачи в проектировании и интеграции программного и аппаратного обеспечения, возникающие при его внедрении, а также возможности применения архитектурных моделей для автоматизации этих задач. Раздел 3 содержит сравнение распространенных языков описания архитектурных моделей. В 4-м разделе представлен краткий обзор инструментальных средств MASIW, разработанных в ИСП РАН, с целью автоматизации проектирования систем ИМА. Следующие два раздела посвящены отдельным исследовательским задачам, возникшим при разработке MASIW. В разделе 5 рассматриваются задачи анализа архитектурных моделей, а в разделе 6 — задачи синтеза отдельных частей модели. В разделе 7 представлено сравнение инструментальных средств MASIW с другими инструментами, построенными на основе языка описания архитектурных моделей AADL. В заключении обсуждаются перспективы развития методов и инструментов моделирования, синтеза и верификации сложных систем на основе архитектурных моделей.

2. Интегрированная модульная авионика

Долгое время доминирующим подходом к построению бортовых электронных систем воздушных судов была так называемая федеративная архитектура авионики, согласно которой каждая подсистема самолета единолично владела полным набором собственного программного и аппаратного обеспечения,

включающего датчики, исполнительные устройства, управляющие контроллеры и провода их соединяющие. Однако к началу 90-х этот подход начал исчерпывать свои ресурсы, поскольку суммарный вес, энергопотребление и стоимость авионики достигли критических величин (рис. 1) [2], что потребовало внедрения принципиально новых подходов к построению бортовых электронных систем.

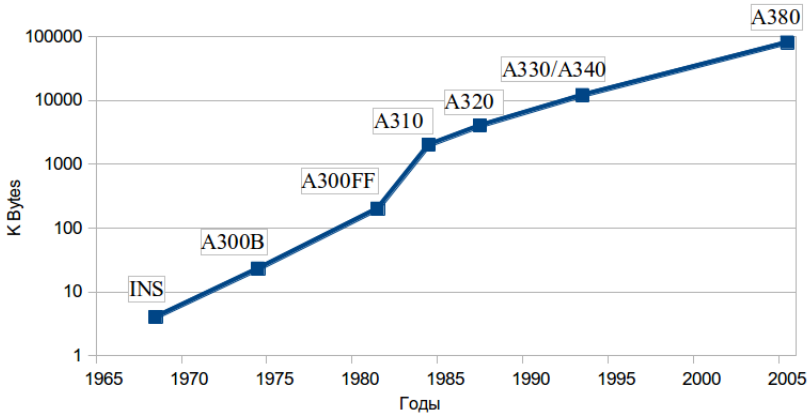

Рисунок 1. Рост размера исходного кода ПО самолетов AIRBUS

В настоящее время основным подходом к проектированию и разработке бортовых систем гражданских воздушных судов является подход интегрированной модульной авионики. Согласно этому подходу, специализированные контроллеры заменяются на процессорные модули общего назначения, на которых обеспечивается независимая работа различных авиационных систем, собственные провода каждой авиационной подсистемы заменяются на виртуальные соединения внутри коммутируемой сетевой инфраструктуры, основанной на таких технологиях, как AFDX (Avionics Full DupleX Switched Ethernet) [3][4][5] и CAN (Controller area network) [6][7]. Это позволяет снизить необоснованное дублирование аппаратного обеспечения, приводящее к неприемлемому уровню энергопотребления и сложности системы бортового оборудования. Но, с другой стороны, такой подход в значительной мере усложняет процесс разработки программного и аппаратного обеспечения, ставит новые задачи в проектировании и интеграции программного и аппаратного обеспечения.

При внедрении подхода ИМА в комплексе бортового оборудования воздушного судна появляется новая подсистема, предоставляющая аппаратную платформу для работы программного обеспечения других бортовых систем. Эта подсистема получила название платформы ИМА и кодовое обозначение ATA-42. Команда, ответственная за проектирование, конфигурирование и верификацию платформы ИМА, обычно называется

Группой системной интеграции, поскольку её задачей является не только разработка отдельно стоящей подсистемы, но и согласование потребностей всех пользователей платформы и конечная интеграция всего комплекса программных и аппаратных компонентов, работающих с использованием платформы ИМА.

В число задач Группы системной интеграции также входит:

- уточнение/согласование требований/потребностей с разработчиками программного и аппаратного обеспечения;
- проектирование платформы ИМА исходя из потребностей функциональных приложений в аппаратных ресурсах, в том числе:
  a. распределение функциональных приложений по вычислительными модулям (Core Processing Module – CPM) с учетом потребностей приложений (количество процессорного времени, распределение процессорного времени между строго периодическими приложениями, объем памяти ОЗУ/ПЗУ, пропускная способность сетевых интерфейсов и т. п.);
  b. определение состава сетевых компонентов (топологии сети) с учетом требований надежности, времени доставки сообщений от отправителя к получателю и т. п.
- проверка разрабатываемого комплекса бортового оборудования (КБО) на соответствие требованиям, изложенных в проектной документации к самолету, КБО и его отдельным компонентам;
- подготовка конфигурационных таблиц для компонентов платформы ИМА.

Для решения этих задач требуется точное понимание всех деталей разрабатываемого комплекса как на высоком, так и на низком уровне детализации, а также предельная внимательность при анализе последствий, в случае внесения изменений. При этом размер КБО современных воздушных судов и количество существенных деталей таково, что их невозможно удержать в голове одного человека. В таких условиях применение специалистами традиционных способов разработки, основанных на аккуратном описании всех требований, архитектурных решений и т. п. в текстовых документах, становится чрезмерно трудоемким и подверженным ошибкам. Возможность подключить к разрешению этих проблем программные средства автоматизации наталкивается на разнородность и неструктурированность информации. Естественным шагом по преодолению этой проблемы является формализация информации, переведение её в унифицированный машино-читаемый вид, что позволяет автоматизировать её обработку.

Рисунок 2. Место ранней валидации в процессе проектирования и разработки платформы ИМА

В контексте проектирования комплексных программно-аппаратных систем, таких как платформа ИМА, основным стержнем является архитектура комплекса, вокруг которой выстраиваются требования к системе в целом и к её отдельным компонентам, проектные компромиссы, работы по анализу и верификации и т. д. Поэтому неудивительно, что именно архитектурные модели, описывающие компоненты системы и связи между ними, становятся основой для формирования новых технологий и инструментов автоматизации проектирования. Они позволяют описывать различные аспекты архитектуры в единой формализованной модели, которая может обрабатываться различными инструментами с целью проверки внутренней согласованности архитектуры, удовлетворения системой разнообразных требований к ней, автоматизации принятия проектных решений, генерации конфигурационных данных/файлов, исходных текстов и т. п. При этом инструменты анализа модели могут быть применимы на разных уровнях абстракции, в том числе на самых ранних стадиях проекта в условиях наличия только частичной и оценочной информации. Среди специалистов данная практика получила название «ранняя валидация» («Early Validation»), а наборы соответствующих инструментов – «инструменты ранней валидации» («Early Validation Tools») [8].

Место применения таких инструментов в процессе проектирования и разработки платформы ИМА показано на рис. 2. Использование архитектурных моделей в этой области позволяет решать следующие задачи:

1. Проверка ограничений/требований предъявляемых к компонентам разрабатываемого комплекса:
   a) проверка достаточности аппаратных ресурсов, например, что потребности всех функциональных приложений в процессорном времени и объеме памяти соответствуют аппаратным характеристикам вычислительного модуля, на котором данные функциональные приложения будут выполнятся;
   b) проверка временны*х* характеристик взаимодействия функциональных приложений или вычислительных модулей, например, что время доставки сообщения от одного функционального приложения до другого не превышает заданного требованиями;
   c) проверка возможности выделения того или иного аппаратного ресурса в соответствии с определенными ограничениями, например, возможности выделения процессорного времени набору строго периодических задач с учетом того, что каждая задача должна запускаться в определенные моменты времени в соответствии с заданным периодом;
   d) проверка безопасности и устойчивости к отказам отдельных компонентов КБО (safety analysis).
2. Автоматизация распределения аппаратных ресурсов между функциональными приложениями с учетом заданных ограничений, например, распределение функциональных приложений по вычислительным модулям с учетом достаточности пропускной способности сетевых интерфейсов и возможности построения расписания для строго периодических задач.
3. Генерация элементов платформы КБО: конфигурационных данных/файлов, исходных кодов отдельных компонентов платформы и т. п.

3. Языки описания архитектурных моделей

За время исследований в области проектирования программно-аппаратных систем на основе моделей сформировалось несколько подходов к описанию архитектурных моделей. Наиболее широкое распространение получили подходы, основанные на языках AADL [9], EAST-ADL [10] и UML [11]. Язык EAST-ADL в настоящей работе не рассматривается, так как его область применения ограничена автомобильными системами, основанными на архитектурных решениях AUTOSAR. Язык AADL унаследовал основные черты от языка Meta-H [12], разработанного для описания бортовых систем авионики в конце 90-х, а в настоящее время является распространенным

языком описания архитектурных моделей программно-аппаратных систем в различных прикладных областях. Язык UML чаще всего применяется для описания программно-аппаратных систем в виде одного из своих профилей, наиболее востребованными среди которых являются SysML [13] и MARTE [14]. Ниже представлены основные особенности данных языков [15].

UML	**AADL**
Нотации	
• предоставляет набор диаграмм для представления структуры программного обеспечения; при этом отдельные диаграммы, описывающие те или иные компоненты программно-аппаратного комплекса, не могут быть полностью связаны друг с другом, т. е. объединение моделей разработанных разными группами разработчиков крайне затруднительно	• разработан более в традициях языков программирования, нежели описания диаграмм; он оперирует декларациями типов и реализаций компонентов модели, которые могут быть переиспользованы в декларациях других компонентов.
Расширяемость	
• может быть расширен путем использования следующих механизмов: ○ стереотипов (stereotypes), которые позволяют расширять словарь UML для создания новых элементов моделирования; ○ картами соответствия идентификаторов и значений (tagged values); ○ переопределение модельных элементов при помощи дополнительных ограничений (constraints); ○ эти механизмы обычно используются тем или иным профилем, который представляет из себя диалект описания моделей (например, SysML и MARTE).	• может быть расширен путем определения: ○ определяемых пользователем наборов свойств, которые могут добавлять новые типы и определения свойств или расширять уже имеющиеся типы и свойства; ○ Annex-спецификаций, которые позволяют описывать дополнительные характеристики элементов модели в произвольном синтаксисе и с произвольной семантикой, которая обрабатываются специализированными инструментами;

Аспекты моделирования	
• используется в основном для описания структуры программного обеспечения; при этом базируется на трех аспектах: данные, взаимодействие и состояния; данные описываются диаграммами классов, взаимодействие описывается диаграммами соединений или диаграммами последовательностей, состояния описываются диаграммами состояний. Наиболее используемые профили SysML и MARTE расширяют UML следующим образом: ○ SysML добавляет два вида диаграмм – диаграмму требований и параметрическую диаграмму; диаграмма требований используется для описаний требований и связи требований с элементами модели; параметрическая диаграмма используется для описания взаимосвязей компонентов модели программного обеспечения с компонентами модели аппаратного обеспечения. ○ MARTE расширяет UML посредством введения следующих стереотипов: модель программного обеспечения, модель аппаратного обеспечения, взаимосвязь моделей программного и аппаратного обеспечения.	• используется для описания «архитектуры исполнения». «Архитектура исполнения» неявно подразделяется на две части: набор программных компонентов и взаимодействие между ними, набор компонентов аппаратного обеспечения и взаимодействие между ними; также описывается взаимосвязи между программными компонентами и компонентами аппаратного обеспечения.

Исходя из вышеперечисленного, можно сделать вывод, что и UML (в виде профилей SysML и MARTE), и AADL предоставляют примерно одинаковые возможности по описанию программно-аппаратной модели КБО. В то же время AADL обладает рядом преимуществ:

- AADL помимо графической нотации имеет текстовое представление, которое позволит специалисту создавать и редактировать модели, а так же анализировать семантику существующих моделей без наличия

специализированных редакторов, в то время как «чтение» моделей на базе UML без использования специальных редакторов диаграмм может стать неразрешимой задачей;

- AADL ограничивает разработчика конкретным набором типов деклараций (типов элементов модели), обладающих определенной семантикой, которые разработчик может использовать для описания модели программно-аппаратного комплекса, что позволяет без дополнительных затрат переиспользовать существующие модели, разработанные независимыми командами. В то же время UML из-за своей универсальности не накладывает строгих ограничений на типы и семантику используемых элементов, что затрудняет понимание моделей, разработанных сторонними специалистами.

4. MASIW – рабочее место системного интегратора

Ввиду рассмотренных выше особенностей язык AADL был выбран в качестве формализма для описания архитектурных моделей в рамках исследований в области автоматизации проектирования программно-аппаратных систем. Одновременно с этими исследованиями в сотрудничестве с коллективом ГосНИИАС ведется разработка инструментальных средств поддержки проектирования ИМА, которые получили название MASIW (Modular Avionics System Integrator Workplace).

В рамках исследований преследуется двуединая цель, состоящая из исследовательской составляющей – развития методов моделирования и верификации комплексных программно-аппаратных систем, и инженерной составляющей – разработки рабочего инструментария для проектировщиков и интеграторов систем авионики.

Основные принципы, на которых построены исследования и инструменты, заключаются в следующем:

- открытость – как необходимое условие сотрудничества с международным исследовательским сообществом;
- опора на международные стандарты;
- сочетание математической строгости в выборе предложенных решений и обеспечении доступности этих решений для инженеров;
- нацеленность на поддержку и интеграцию различных процессов жизненного цикла работки систем: определения и анализа требований, проектирования, интеграции и верификации программных и программно-аппаратных систем.

На текущий момент разработанные инструментальные средства MASIW позволяют решать следующие задачи.

1. Создание, редактирование и управление моделями на языке AADL:
   a) создание/редактирование моделей посредством текстового или графического редактора;

b) поддержка командной разработки с возможностью отслеживания и внесения изменений для отдельных элементов модели;
c) поддержка переиспользования AADL моделей сторонних разработчиков.

2. Анализ моделей:
   a) анализ структуры программно-аппаратного комплекса – достаточности аппаратных ресурсов, согласованности интерфейсов и т. п.;
   b) анализ характеристик передачи данных в сети AFDX – времени доставки сообщений от отправителя к получателю, глубины очередей передающих портов и т. п.;
   c) симуляцию модели программно-аппаратного комплекса с генерацией пользовательских отчетов по результатам работы симулятора.
3. Синтез моделей:
   a) распределение функциональных приложений по вычислительным модулям с учетом ограничений ресурсов аппаратной платформы и с учетом дополнительных ограничений касающихся вопросов надежности и безопасности программно-аппаратного комплекса;
   b) генерация распределения вычислительного времени процессора между функциональными приложениями (циклограмма расписания запуска приложений для ARINC-653 совместимых операционных систем реального времени);
4. Генерация исходного кода/конфигурационных данных:
   a) разработка специализированных инструментов генерации кода/конфигурационных данных, на основе предоставляемого программного интерфейса (API);
   b) генерация конфигурационных файлов для ОС РВ VxWorks653 и оконечных устройств сети AFDX.

Создание, редактирование и управление моделями, а также генерация кода и конфигурационных данных реализованы с использованием широко распространенных расширений среды Eclipse, такими как Eclipse Modeling Framework, Graphical Editing Framework, Eclipse Team Providing, SVN Team Provider, GIT Team Provider. При реализации этих возможностей в основном приходилось решать инженерные задачи, поэтому в последующих разделах мы более подробно остановимся на реализациях поддержки анализа и синтеза моделей, где были сконцентрированы основные исследовательские задачи.

5. Анализ моделей

Когда заходит речь об анализе моделей, то под этим понимают выведение новых свойств модели в результате рассуждений об ее уже известных свойствах. Например, результатом анализа может быть оценка наибольшего времени между отправкой сообщения и его доставкой на основании анализа

пути следования сообщения и характеристик компонентов, встречающихся на этом пути. Наиболее важным видом анализа модели является ее верификация, то есть проверка выполнения моделью требований, которые к ней предъявляются. Другие виды анализа, как правило, используются в качестве промежуточного шага в процессе верификации.

Требования к архитектуре КБО возникают из разнообразных источников.

- Это могут быть проектные требования к самолёту и архитектуре КБО — эти требования в процессе анализа уточняются и декомпозируются на требования к отдельным компонентам системы.
- Часто в рамках определенного проекта регламентируются требования на оформление и организацию архитектурных моделей, которые описываются в так называемом стандарте на проектирование моделей.
- Еще одним источником требований являются ограничения на область допустимого использования или на допустимые конфигурации моделируемых компонентов (usage domain rules).
- Автор библиотечного модельного компонента может накладывать требования по консистентному использованию этого компонента.
- Также встречаются требования, накладываемые инструментами или средствами анализа моделей и необходимые для возможности проведения соответствующего анализа.

Так как при моделировании системы есть потребность обнаруживать ошибки как можно раньше, то стоит задача анализа модели, в которой есть недоспецифицированные компоненты или компоненты с еще неизвестной структурой. Иногда в таких случаях для проведения некоторого вида анализа достаточно предположений о неразработанном компоненте. Например, в системе есть процесс A с неизвестной реализацией. Однако, предполагается или известно, что в среднем он раз в 100 мс генерирует пакет данных размером в среднем 100 байт, предназначенный процессу B. При этом компоненты, обеспечивающие сетевое взаимодействие, описаны в модели достаточно детально. Тогда такую неполную модель можно анализировать в аспекте сетевого взаимодействия, задержек при связи процессов, заполненности буферов сетевых компонентов и пр.

5.1. Типы анализа моделей

Типы анализа различаются по *способу* его выполнения (**статический** или **динамический**) и по *аспектам* исследуемого объекта (самое грубое деление это аспекты **структуры** или архитектуры системы и аспекты **поведения**, функционирования системы).

Динамический анализ подразумевает, что явным образом заданы некоторые закономерности, по которым происходит изменение модели (внутреннего состояния компонентов, связей между ними) и взаимодействие с окружающей средой. Во время проведения такого анализа происходит выполнение

действий по заданным закономерностям, получение новых состояний компонентов, новых связей, обеспечение взаимодействия с окружающей средой. В дальнейшем (в зависимости от проверяемых требований) происходит анализ полученного состояния или последовательности состояний и, например, оценка их корректности.

При **статическом анализе** используется математическое описание компонентов модели, которое сопоставляется с описанием требований на них. В процессе анализа производится сопоставление требований, вычисление характеристик компонентов, по которым в дальнейшем делаются выводы о корректности или некорректности анализируемой модели.

При анализе **поведения** модели рассматриваются характеристики, возникающие только при рассмотрении того, как себя ведут компоненты во времени, как они взаимодействуют с окружающим миром, на какие события и как реагируют, какие события и данные генерируют, как изменяют своё внутреннее состояние. При анализе **структуры** рассматриваются характеристики того, как соединены компоненты, какими свойствами обладают эти связи, какие есть возможности и направления передачи данных и событий, у каких компонентов есть доступ до тех или иных ресурсов и пр.

Способы и аспекты моделирования могут сочетаться произвольным образом, далее, в разделах 5.3-5.6, рассматриваются все четыре возможные комбинации.

5.2. Входные данные для анализа

Входными данными для анализа является модель программно-аппаратного комплекса, которая описывает структуру и характеристики/свойства элементов комплекса. Рассмотренные ранее языки описания моделей (UML и AADL) позволяют детально описывать структуру разрабатываемого комплекса, вплоть до описания каждого датчика, кнопки и т.п. Как показала практика, такая модель является избыточной для большинства видов анализа. Значительная часть модели игнорируется специализированными инструментами анализа, и в результате инструменту приходится выполнять лишнюю работу по выборке существенных данных из модели комплекса. Кроме того, языки описания моделей предоставляют разработчику моделей определенную свободу в выборе того, какими сущностями будут описаны те или иные компоненты разрабатываемого комплекса (UML в большей степени, AADL – в меньшей). В то же время при разработке инструмента анализа можно точно определить структуру входных данных, которая не зависит от того, какими конкретными сущностями описана модель программно-аппаратного комплекса. Поэтому при разработке инструмента MASIW была предложена концепция так называемых представлений. **Представление** (англ. view) – специализированная модель всего программно-аппаратного комплекса или определенной его части, которая представляет набор существенных данных в виде удобном для обработки, как это происходит с представлениями

в системах управления базами данных. Для создания специализированного представления применяется набор адаптеров – правил, по которым исходная модель трансформируется в специализированную и, при необходимости, наоборот. Такой подход позволяет абстрагироваться от того, каким образом разработчик опишет модель комплекса или его части (какие сущности будут применены и даже какой язык описания моделей будет использован). Благодаря этому разрабатываемые инструменты анализа могут быть переиспользованы в других инструментах работы с моделями.

5.3. Статический структурный анализ

Структуру модели можно понимать как граф, узлами которого являются компоненты модели, а дуги - связи между компонентами. Виды связей могут отличаться в разных языках моделирования. Например, связь между двумя компонентами может означать, что один компонент является частью другого, или что один компонент является аппаратным ресурсом, на котором выполняется другой, программный компонент.

Структура модели может содержать информацию о составе компонентов модели (из каких компонентов-частей он состоит), о размещении компонентов, о степени связности компонентов и т.п. Зачастую языки моделирования позволяют сопоставить компонентам модели атрибуты, т.е. некоторые скалярные значения. В этом случае они также становятся частью структуры модели и могут быть использованы в структурном анализе. Например, можно проанализировать модель на предмет того, *принадлежат ли значения атрибута с некоторым заданным именем во всех компонентах, у которых он определен, некоторому заданному множеству допустимых значений*.

Одним из возможных атрибутов компонентов может быть тип компонента. Примерами использования типа в структурном анализе могут быть следующие задачи: *узнать, содержат ли все компоненты некоторого типа атрибут с некоторым заданным именем*. Или другой пример: *выяснить, все ли компоненты типа A являются частью какого-либо из компонентов типа B*.

В тех языках моделирования, которые нацелены на описание структуры модели (в отличие от нацеленных на поведение), структурный анализ модели удобнее, чем поведенческий. Причина этого в том, что при использовании таких языков моделирования структура модели уже может присутствовать, в то время как поведение еще до конца не определено или не описано. Как следствие, ряд анализов можно провести заранее, до более трудоемкой операции определения поведенческой составляющей модели.

Для организации и автоматизации статического структурного анализа необходимо решать следующие задачи:

1. как задать то, *что* надо проанализировать (в частности, какое условие корректности структуры модели необходимо проверить и какой язык

следует выбрать для описания этого условия);

2. как задать *контекст* анализа (для какой части структуры модели необходимо проводить анализ) - чаще всего, анализ следует проводить на всей модели, хотя вполне возможно, например, что анализ надо проводить лишь на компонентах, являющихся частью данного компонента модели, или на компонентах лишь некоторого заданного типа;
3. как *выполнять* заданный анализ для заданной части модели;
4. в каком *виде* представить разработчику результат анализа.

Все эти задачи возникли и при разработке инструмента статической проверки правильности структуры моделей в среде MASIW. В этой среде в качестве языка моделирования используется AADL. Структура модели в этом языке является иерархической по отношению включения одних компонентов в другие. Также модель на языке AADL содержит иерархию типов, описывающих классы компонентов. Среда MASIW строит экземпляр модели, генерируя экземпляры всех компонентов и соединяя эти компоненты так, как того требует семантика языка AADL с учетом всех наследуемых и переопределяемых атрибутов. В результате инструменты анализа работают с уже подготовленным экземпляром модели и для них нет необходимости знать о сложностях трансформации декларативной AADL модели в экземпляр архитектурной модели программно-аппаратного комплекса.

В соответствии с контекстом проведения анализа обычно выделяют глобальные условия корректности и компонентные условия корректности. Глобальные условия корректности представляют ограничения на модель в целом. Компонентные условия корректности описывают ограничения на определенные виды компонентов. Иногда про компонентные условия говорят как об “инвариантных свойствах компонентов”.

Чтобы иметь возможность автоматически проводить анализ модели на выполнение требований, требования должны быть формализованы и описаны в машинно-читаемом виде. Для этого необходимо иметь язык для записи формализованных требований. В настоящее время в MASIW поддерживается описание и проверка условий корректности архитектурной модели на языке REAL (Requirements Enforcement Analysis Language) [16].

Язык REAL был предложен в 2010 году в Telecom ParisTech (Франция) и был поддержан сразу несколькими исследовательскими лабораториями по всему миру. Этот язык основан на аппарате теории множеств. Автор языка пытался сделать язык как можно более удобным для инженеров, занимающихся моделированием и владеющих базовыми навыками императивного программирования. Однако этот язык оказался непригодным для практического использования ввиду своей ограниченной функциональности, а в открытой печати так и не появилась качественная документация по этому языку.

После этого в Rockwell Collins был разработан язык Lute [17] и инструмент по проверке моделей на выполнение требований на Lute. Язык Lute является наследником языка REAL, незначительно расширяя его синтаксис. Кроме того, вместе с инструментом поставляется библиотека теорем, разработанная в Rockwell Collins в рамках проекта META. Однако качественная документация по языку Lute и библиотеке теорем не доступна, что не дает возможности оценить выразительные возможности этого языка.

Как показала практика язык REAL содержит ряд серьезных недостатков. Во-первых, он поддерживает не все типы данных и компонентов языка AADL (в частности, он совсем не поддерживает значения с единицами измерения). Во-вторых, он не поддерживает компонентные условия корректности. В-третьих, в языке отсутствуют средства переиспользования части одних условий в других (что особенно актуально, если проверка нескольких условий требует одних и тех же вычислений). Кроме того, не все условия корректности удобно представлять в императивном виде, а язык REAL не содержит средств для неимперативного описания условий корректности.

Для частичного решения этих проблем мы внесли ряд изменений в язык REAL. Более того, мы поставили цель сделать статический анализ не столько средством демонстрации правильности модели на AADL, сколько средством поиска ошибок в модели. Для этого мы, формально не меняя язык, предложили способ аннотирования текста утверждения на REAL с целью документирования его семантики. Эта документация используется нашим инструментом при построении отчёта о проведенной верификации, чтобы продемонстрировать, какие условия корректности проверялись, на каких компонентах были проверены условия корректности, каков статус проверки на компонентах (на каких компонентах обнаружено нарушение условий корректности, а на каких - нет), причина нарушения (если она была задана в комментариях-аннотациях).

5.4. Статический поведенческий анализ

Целью статического анализа поведения модели системы является получение математическими методами оценок предельных величин различных характеристик поведения и взаимодействия компонентов системы.

Одной из важнейших поведенческих характеристик является время реакции системы на поступающие извне события. На время реакции влияет как скорость обработки событий, так и время доставки информации между компонентами. В основе архитектуры ИМА лежит идея разделения аппаратных ресурсов между множеством авиационных функций с обеспечением отсутствия непреднамеренного влияния одной функции на другую. Первым шагом в этом направлении было разделение вычислительных ресурсов. Следующим шагом стала виртуализация шин данных, которая начиная с Airbus-380 базируется на технологии AFDX [5].

AFDX построена на основе обычного Ethternet, но доработана для обеспечения детерминированности, устойчивости, безопасности, надежности, необходимых для удовлетворения требований сертификации. Ключевым элементом AFDX в этом плане является понятие виртуального канала (Virtual Link), которое по сути представляет собой виртуальный провод, эквивалентный физическому проводу между отправителем и получателем сообщений. Виртуальность проводов сокращает вес, энергопотребление, сложность прокладки и, что самое главное, стоимость сопровождения и развития сети, так как прокладка физического кабеля заменяется на модификацию конфигурационных таблиц в коммутаторах.

Собственно компоненты, генерирующие и получающие сообщения, могут находится вне сети AFDX. Это либо специализированные управляющие функции, располагающиеся в датчиках и актуаторах, либо приложения, выполняющиеся на вычислительных модулях. Во всех случаях эти компоненты соединены с сетью AFDX посредством одного или нескольких промежуточных шлюзов, сообщение с которыми может происходить по нескольким разным протоколам передачи данных.

Анализ передачи данных в сети AFDX

Технология AFDX построена на основе полнодуплескного коммутируемого Ethernet, поэтому конфликтов и задержек на линиях при передаче данных не возникает. Общее время доставки пакета равно сумме времен передачи пакета на линиях плюс задержки в коммутаторах.

На выходных портах коммутаторов установлены очереди. Таким образом, задержка в коммутаторах может быть очень изменчивой из-за слияния различных виртуальных каналов, конкурирующих за каждый выходной порт. Следовательно, для определения верхней границы общего времени передачи пакета необходимо анализировать задержки в каждом выходном порту коммутатора.

Еще одной важной поведенческой характеристикой сети AFDX является гарантия того, что в каждой очереди имеется достаточно места для хранения всех прибывающих пакетов. На практике для каждой очереди в сети оценивается верхняя граница количества данных в этой очереди.

Существует несколько аналитических методов для вычисления оценки верхних границ времени доставки пакета и размеров очередей: Model Checking, Trajectory, Network Calculus. Каждый имеет свои преимущества и недостатки. Главным недостатком большинства методов является так называемый пессимизм — то есть получение заведомо большей оценки из-за некоторых предположений или грубых вычислений. Кроме того, существенным недостатком метода Model Checking является то, что при увеличении размера сети он очень быстро приводит к комбинаторному взрыву. Для промышленного использования подходят лишь методы Trajectory и Network Calculus.

Метод Trajectory

Метод Trajectory [18] базируется на анализе сценария худшего случая, который может произойти с пакетом на его траектории. *Интервалом занятости* для пакета f в выходном порту p называется временной интервал, в течение которого f может обрабатываться в p. Метод Trajectory предполагает наибольший интервал занятости в каждом порту. Для каждого канала-конкурента оценивается максимальное количество его пакетов, которые могут задержать отправку пакета f из порта p во время интервала занятости.

При вычислении верхней границы времени доставки пакета для данного виртуального канала, предполагается, что пакет f становится на выходном порту p в очередь, в которой уже стоят по максимальному количеству пакетов всех остальных виртуальных каналов, которые могут задержать отправку пакета f.

Оценка верхних границ размеров очередей методом Trajectory производится так. Необходимо для каждого выходного порта найти максимум из размеров очередей среди интервалов занятости всех виртуальных каналов, отправляемых через этот порт. Это значение и будет верхней границей размера очереди данного порта.

Метод Network Calculus

В методе Network Calculus [19] поток информации через определенный узел сети представляется функцией потока. *Функцией потока* R(t) от времени называется функция, значением которой в момент t является суммарное количество бит, вошедшее в этот узел начиная с момента $t_0=0$.

Поскольку на характер функции потока в данном узле, вообще говоря, влияют много разных факторов, точное ее определение довольно затруднительно. Для анализа потоков информации в методе Network Calculus используются так называемые входящие кривые (arrival curve), которые мажорируют функцию потока “равномерно” начиная с любого момента времени: *входящая кривая* $\alpha(t)$ потока R(t) — это такая неубывающая функция, что для любых $0 \le s \le t$ верно неравенство $R(t) - R(s) \le \alpha(t-s)$.

Если поток информации представляет из себя периодическую последовательность пакетов ограниченной длины, функция R(t) является “ступенчатой” функцией. Входящей кривой для функции такого рода является функция $\gamma_{r,b}(t) = rt + b$, где r – это средняя скорость потока, b – максимальный размер пакета.

Кроме входящей кривой, которая является “представителем” функции потока, в методе Network Calculus в каждом узле сети рассматривается *обслуживающая кривая* (service curve), которая описывает количество обработанной информации в узле к данному моменту времени.

Рассмотрим ситуацию, при которой узел обрабатывает информацию с постоянной скоростью R (как правило, эта скорость соответствует пропускной

способности линии связи на выходе из узла), однако перед обработкой вносит некоторую задержку, ограниченную временем T (как правило, это время соответствует максимальной технологической задержке по доставке информации внутри узла от входа к выходу). В таком случае обслуживающая кривая в этом узле равна $\beta_{R,T}(t) = \max\{ 0, R(t-T) \}$.

Максимальная задержка, которую информация из потока с входящей кривой α получает в узле, который обеспечивает обслуживающую кривую σ, оценивается сверху величиной максимального горизонтального (т. е. по оси «время») отклонения от α до σ. Для случая $\alpha = \gamma_{r,b}$ и $\sigma = \beta_{R,T}$ размер максимальной задержки равен $T + b/R$.

Максимальное количество необработанной информации из потока с входящей кривой α в узле, который обеспечивает обслуживающую кривую σ, оценивается сверху величиной максимального вертикального (т. е. по оси «информация») отклонения от α до σ. Для случая $\alpha = \gamma_{r,b}$ и $\sigma = \beta_{R,T}$ максимальное количество необработанной информации равно $\gamma_{r,b}(T) = b + rT$.

Использование огрубления в виде входящих кривых вместо функций потока приводит к тому, что по мере прохождения пакета по узлам сети входящая кривая становится все более «грубой». В частности, для случая, когда в узле входящая кривая для входного потока равна $\alpha = \gamma_{r,b}$, а обслуживающая кривая равна $\sigma = \beta_{R,T}$, входящая кривая для выходного потока (она же будет являться входящей кривой для входного потока следующего узла на пути данного пакета) равна $\gamma_{r,b+rT}$, т.е. по мере прохождения пакета узлам сети происходит рост второго параметра функции γ, так что на последующих узлах соответственно растут оценки времени задержки и количества необработанной информации.

Если сравнивать Network Calculus и Trajectory между собой, то нельзя сказать об однозначном превосходстве одного из методов. Хотя во многих случаях Trajectory дает более точные оценки наихудшего времени, чем Network Calculus, существуют такие конфигурации сети, на которых наблюдается обратная ситуация.

5.5. Динамический поведенческий анализ

Динамический анализ поведения позволяет получать менее пессимистичные оценки поведенческих характеристик, по сравнению со статическим анализом. Помимо этого, иногда применение динамического анализа позволяет провести анализ в тех случаях, когда статически его провести не получается или это требует слишком много ресурсов (слишком большая сложность модели, трудносоставимое и трудноанализируемое математическое описание компонентов, комбинаторный взрыв и пр.). Однако, надо учитывать, что динамический анализ проводится в конкретном выполнении, а не в худшем случае, и требует задания конкретного контекста работы: входных данных и воздействий.

Для проведения динамического поведенческого анализа требуется, чтобы в модели некоторым исполнимым образом было задано поведение компонентов. Для корректного использования исполнимых моделей важно организовать работу с модельным временем, с передачей информации и событий внутри модели.

Эта задача хорошо решается при использовании дискретно-событийного подхода к моделированию поведений [20]. В этом подходе работа компонента представляется как набор “событий” – актов действий по изменению состояния объектов и взаимодействию с другими компонентами и внешним миром в определённые моменты времени.

Этот подход зарекомендовал себя как наиболее подходящий при моделировании поведений компьютерных систем [21], какими являются системы КБО ИМА. С одной стороны, он является достаточно мощным для моделирования таких систем, с другой стороны, такой подход гораздо более легко применим (в отличие от ещё более мощного подхода непрерывной симуляции [22], сталкивающегося с решением нелинейных дифференциальных уравнений и используемый при моделировании физических процессов).

Для поддержки дискретно-событийной парадигмы описания поведения, в инструменте MASIW была реализована библиотека, поддерживающая симуляционное время и обеспечивающая обработку событий, возникающих с моделируемой системе, и синхронную и асинхронную передачу данных. Для обеспечения этих возможностей был использован подход в стиле продолжений (continuations programming) [23], для поддержки которого была использована библиотека Matthias Mann’s continuations [24].

Язык AADL имеет собственные средства описания поведенческой семантики для некоторых элементов модели. Однако этих средств недостаточно для описания поведения приложений, устройств и других сложных компонентов модели. Существуют стандартизованные расширения AADL — Behavioral Model Annex [25] и BLESS [26], — позволяющие описывать поведение компонентов модели на основе конечных автоматов, работающих со временем и событиями.

На данный момент в инструменте MASIW исполнимая модель поведений задаётся на языке Java. Это позволило довольно быстро реализовать поддержку динамического поведенческого анализа AADL-моделей и в данный момент инструмент уже можно использовать для такого анализа. Однако приходится задавать поведения компонентов нестандартным способом. С другой стороны, использование Java позволит в будущем реализовать трансляторы со стандартных языков задания поведения без необходимости переделывания части, отвечающей собственно за динамический анализ.

Как было сказано ранее, часто целесообразно проводить ранний анализ системы на неполных моделях. В таких моделях некоторые компоненты ещё не имеют детального описания, а часто есть только некоторые предположения

о том, как они себя ведут в этой системе. В таком случае анализ можно производить, записав предположения в виде того или иного поведения компонентов. В инструменте MASIW поддерживается специальный способ параметризации модели, упрощающий описание различных предположений о компонентах для проведения различных вариантов анализа модели.

5.6. Динамический структурный анализ

Этот тип анализа требуется для реконфигурируемых систем, то есть для систем, структура которых может меняться во время работы. В общем случае, может существовать зависимость структуры модели от входных воздействий или окружения моделируемой системы, поэтому не всегда можно статически проанализировать выполненность всех требуемых структурных ограничений.

В чистом виде динамический структурный анализ предполагает проверку того, чтобы все достижимые состояния архитектурной модели являются структурно-корректными, то есть удовлетворяют требованиям на корректность структуры модели. Для проверки свойств структуры модели, полученной в какой-либо момент выполнения модели, требуется получать изменённую структуру и запускать проверки над новой моделью. Такая проверка подобна статической структурной проверке, на вход которой подаётся модель, полученная в динамике. Сложность здесь состоит в том, что не всегда нужно проверять все свойства во всех состояниях, но требуется некоторым образом задавать, какие проверки в какой момент осуществлять.

Динамический структурный анализ может также проверять свойства, которые задаются не над одним состоянием структуры модели, а над последовательностью таких состояний, хотя такой анализ скорее является анализом поведенческого аспекта функциональности по реконфигурированию системы.

Традиционно среди свойств объектов во времени выделяют свойства *безопасности* и свойства *живучести* [27]. Первые требуют, чтобы что-то никогда не случилось, в то время как вторые — чтобы что-то когда-нибудь обязательно случилось. Примером свойства безопасности является требование, чтобы сразу после возникновения события X у заданного компонента A появится подкомпонент B, а свойства живучести — требование о том, что после возникновения события X у заданного компонента A рано или поздно появится подкомпонент B.

Уже рассмотренное требование структурной корректности всех достижимых состояний является простейшим примером свойства безопасности. Динамическая проверка более сложных свойств безопасности может быть реализована по схожим принципам, с учетом того, что часть информации используемой при статическом структурном анализе фиксированного состояния архитектурной модели должна вычисляться на основании свойств предыдущих состояний модели.

Проверка свойств живучести – существенно иная задача. Основной особенностью такого рода свойств является то, что они нарушаются только на бесконечности. Поэтому задача проверки таких свойств не может быть решена чистым динамическим анализом и требует разработки специальных средств.

6. Автоматический синтез моделей

Перед проектировщиком системы ИМА стоит задача построения архитектуры, которая должна удовлетворять требованиям различных видов: по достаточности аппаратных ресурсов, по устойчивости к сбоям, надежности, безопасности системы в целом, ограничениям на максимально допустимое время доставки сообщений между компонентами, требованиям на своевременное выполнение функциями своих действий и т. д.

До определенного уровня решить такую задачу позволяет искусство опытных специалистов, вооруженных кроме того инструментами верификации строящейся архитектурной модели. Однако этот подход имеет ограниченную масштабируемость и высокую субъективность. Средства автоматизации проектирования систем, удовлетворяющих заданным наборам требований и ограничений, могут сделать работу проектировщиков намного эффективнее.

Во многих случаях отдельные части модели могут быть автоматически синтезированы на основании информации, содержащейся в другой («исходной») части модели, которая описывает основные логические связи между компонентами и требования к результирующей архитектуре. При этом, разработка исходной части модели гораздо проще, чем разработка соответствующей синтезируемой части. Кроме того, исходная часть в любом случае должна быть описана в процессе проектирования. Например, на основании исходной информации об имеющемся наборе приложений и их потребностях в аппаратных ресурсах, а также информации об архитектуре и возможностях вычислительных модулей возможно автоматически синтезировать привязку приложений к этим модулям, так чтобы удовлетворить всем требованиям по достаточности ресурсов и возможности построения расписаний.

В среде проектирования MASIW предлагается следующий сценарий работы для разработки модели проектируемой системы. Проектировщик разрабатывает необходимую исходную часть модели, после чего запускает алгоритм автоматического синтеза, который на основании имеющейся информации, содержащейся в исходной части модели, достраивает модель архитектуры новыми частями, которые при необходимости могут быть скорректированы вручную или перегенерированы в случае обновления исходной части модели.

6.1. Автоматический синтез расписаний выполнения строго периодических задач

При разделении аппаратных ресурсов между несколькими приложениями одним из важнейших аспектов является своевременное обеспечение процессорными ресурсами всех выполняющихся в системе задач. Этим аспектом обычно занимается специальная подсистема планирования задач операционной системы, которая выделяет процессорное время функциональным приложениям на основании заранее подготовленного расписания.

В качестве исходных данных в задаче построения расписания для каждой из периодических задач заданы:

- период запуска задачи;
- время выполнения задачи на одном периоде запуска.

Классические алгоритмы планирования периодических задач работают лишь в случае, когда время запуска задачи внутри периода разрешается варьировать на разных периодах ее выполнения. Однако, в настоящее время имеется потребность в составлении таких расписаний, в которых время между соседними запусками одной периодической задачи было бы фиксировано и равнялось бы длине периода. Такое дополнительное требование *строгой периодичности* не позволяет использовать в планировщике классические алгоритмы планирования.

Основной сложностью алгоритма планирования строго периодических задач является поиск стартовых точек для всех задач, так чтобы было возможно построение собственно расписания. Этот поиск является NP-полной задачей.

Предложенный нами алгоритм [28] позволяет сократить поиск, отсекая на ранних стадиях те ветви перебора, которые заведомо ведут к отрицательному результату. Алгоритм перебора основан на теоретико-числовых соображениях. Основной идеей является перебор для каждой следующей рассматриваемой задачи лишь тех точек, которые заведомо не приведут к совпадению стартовых точек данной и никакой из ранее рассмотренных задач. А именно, если уже выбраны стартовые точки r_1, ..., r_k для первых k задач с периодами p_1, ..., p_k соответственно, то при поиске стартовой точки r для очередной задачи с периодом p надо перебирать лишь те точки, которые не удовлетворяют ни одному из сравнений

$$r - r_i \equiv 0 \pmod{\text{НОД}(p_i, p)}$$

для всех i от 1 до k.

Кроме того, используемая нами стратегия поиска стартовых точек подразумевает перебор в первую очередь таких вариантов, которые обеспечивают как можно более длительное непрерывное выполнение первых тиков после запуска каждой задачи.

В целом, такой подход позволяет быстро получать решение задачи построения расписания для строго-периодичных задач.

6.2. Автоматический синтез архитектуры системы ИМА

В качестве исходных данных в задаче синтеза архитектуры ИМА заданы:

- функциональные приложения и логические потоки данных между ними, а также между приложениями и датчиками/актуаторами;
- набор потребностей функциональных приложений к аппаратным ресурсам (память, вычислительная мощность и т.д.);
- набор требований на максимальное время доставки/обработки сообщений в логических потоках данных;
- набор имеющихся аппаратных компонентов (вычислительные модули, коммутаторы и пр.) в совокупности с описанием предоставляемых ими возможностями и ограничениями на область их допустимого использования (usage domain rules).

Требуется автоматически построить архитектуру системы ИМА, включающую в себя:

- состав и связи аппаратных компонентов;
- размещение функций по вычислительных модулям;
- детали организации соединений в AFDX-сети;
- расписание работы прикладных и системных разделов ARINC-653 совместимых операционных систем;

При этом архитектура системы должна удовлетворять всем требованиям по безопасности и производительности.

Задача синтеза разбивается на две крупные подзадачи:

1. размещение приложений по вычислительным модулям так, чтобы на каждом модуле было возможно построение расписания;
2. назначение виртуальных каналов между вычислительными модулями и распределение коммутаторов по виртуальным каналам так, чтобы удовлетворить требованиям по времени доставки сообщений.

Решение первой задачи основано на рассмотрении множества периодов запуска приложений и на применении теоретико-числовых соображений, которые позволяют разбить множества периодов на такие подмножества, что для каждого полученного подмножества гарантированно отыщутся стартовые точки соответствующих приложений.

Решение второй задачи основано на применении генетических алгоритмов. На каждом шаге генетического алгоритма строится популяция, состоящая из N корректных топологий AFDX-сети. Каждая топология новой популяции получается либо в результате небольшой модификации (мутации) некоторой топологии предыдущей популяции, либо в результате скрещивания некоторых

двух топологий предыдущей популяции. При скрещивании результирующая топология получает максимальное количество общих свойств (в смысле соединения компонентов между собой), имеющихся в обеих исходных топологиях.

После построения очередной популяции производится ранжирование входящих в нее топологий с тем расчетом, чтобы для последующего построения выбрать N топологий, наиболее удовлетворяющих требованиям по времени доставки сообщений. Для оценки получившихся в данной топологии времен доставки используются статические методы (Trajectory, Network Calculus), и основная составляющая функции ранжирования выглядит так:

$$\Sigma\, e^{T-\tau},$$

где суммирование ведется по всем каналам, для которых задано ограничение по времени доставки, T — получившееся время доставки для данного канала в данной топологии, τ — заданное максимальное время доставки для данного канала.

7. Сравнение возможностей MASIW с другими инструментами

Язык AADL поддерживается такими инструментами как OSATE (и производными), Ocarina, STOOD/AADL Inspector, а также инструментом MASIW. Возможности этих инструментов представлены в таблице 1.

	MASIW	**OSATE**	**RC_META**	**Adele**	**Ocarina**	**AADL Inspector**
AADL процессор	MASIW		OSATE		Ocarina	STOOD
Наличие графической среды IDE	+	+	+	+	–	+
Текстовый редактор AADL-моделей	+	+	+	+	–	+
Графический редактор AADL-моделей	+	–	∓(1)	+(2)	–	+
Статический структурный анализ AADL-моделей	REAL+	Lute	Lute	–	REAL	Prolog
Статический поведенческий анализ AADL-моделей	см. п. 5.4	–	AGREE(3)	–	–	Cheddar(4)
Динамический структурный анализ AADL-моделей	–	–	–	–	–	–

	MASIW	OSATE	RC_META	Adele	Ocarina	AADL Inspector
Динамический поведенческий анализ	см. п. 5.5	–	–	–	–	∓(5)
Анализ безопасности (Safety Analysis, FMEA)	–	+	–	–	–	–
Возможность переиспользования AADL-библиотек сторонних разработчиков	+	–	–	–	+	+
Специализированные задачи для систем ИМА:						
Генерация кода/конфигурационных файлов	±(6)	±(7)	–	–	±(8)	–
Генерация распределения процессорного времени приложений	+	–	–	–	+	+

(1) RC_META предполагает использование коммерческого инструмента Enterprise Architect для редактирования SysML диаграмм, которые настроены для однозначного представления AADL моделей и автоматически трансформируются в AADL.

(2) Графический редактор Adele поддерживает только старую версию стандарта AADL 1.0.

(3) AGREE (Assume Guarantee Reasoning Environment) позволяет описывать предположения о значениях входных данных компонентов и требования к значениям выходных данных, а также проверять их согласованность [29].

(4) Cheddar — инструмент анализа возможности построения расписаний для различных стратегий планирования [30].

(5) Поддерживается только симуляция распределения процессорного времени при помощи инструмента Marzhin.

(6) Генерация набора конфигурационных файлов для ОС VxWorks653, программного обеспечения промежуточного слоя и оконечных систем сети AFDX.

(7) Проект RAMSES поддерживает генерацию кода на основе подмножества AADL-BA и предварительной трансформации абстрактной архитектурной модели в более детальную [31].

(8) Ocarina поддерживает генерацию кода на языке Си для операционных систем RT-POSIX, Xenomai, RTEMS и на языке Ада для профиля Ravenscar.

Следует отметить, что благодаря плагину, реализующему интеграцию инструмента Ocarina в среду OSATE, в ней появилась поддержка возможностей Ocarina для тех AADL моделей, которые не используют конструкции не поддерживаемые Ocarina.

Основными отличиями инструмента MASIW является то, что, с одной стороны, он представляет удобную интегрированную среду разработки с возможностью создания/редактирования моделей КБО как в текстовом, так и в графическом представлении и базовыми возможностями для проведения статического и динамического анализа, а, с другой стороны, его внутренняя архитектура позволяет расширять его функциональность при помощи компонентов (плагинов к специализированному «рабочему пространству»), созданных сторонними разработчиками.

8. Перспективы развития

На текущий момент времени инструмент MASIW позволяет выполнять лишь часть задач возложенных на группу системной интеграции и в дальнейшем планируется расширять функциональность его функциональность по многим направлениям.

В контексте *статического структурного анализа моделей* основным направлением развития является разработка полнофункционального языка описания ограничений на структуру архитектурной модели удобного для компактного описания как глобальных, так и компонентных ограничений. По нашему мнению, этот язык должен базироваться на одном из хорошо известных существующих языков программирования, чтобы получить возможность переиспользовать уже готовые библиотеки с разнообразной функциональностью и упростить задачу подготовки инженерных кадров. Хорошим претендентом на роль такого языка является язык Python, который за счет концепции декораторов предоставляет возможность сформировать на своей основе специализированный язык, оставаясь в рамках стандартного синтаксиса, что означает возможность использования в неизменном виде для нового языка уже существующего интерпретатора и других инструментов. Другими перспективными направлениями являются развитие библиотек готовых частей кода для их переиспользования при проверке условий корректности и реализация статического структурного анализа реконфигурируемых систем.

В контексте *статического поведенческого анализа моделей* перспективным направлением развития поддерживаемых методов анализа является анализ передачи данных в системе в целом, а не только в рамках сети AFDX. Основную сложность здесь составляет учет особенностей поведения всех

компонентов-шлюзов, располагающихся между отправителем/получателем сообщения и сетью AFDX.

В контексте *динамического поведенческого анализа моделей* основным направлением развития является поддержка стандартных способов задания поведения для компонентов модели (Behavioral Model Annex, BLESS). Другим очень важным направлением развития этого типа анализа является реализация возможности использования симулятора в комплексе со стендом полунатурного моделирования и в комплексе с внешними эмуляторами аппаратных платформ. Это позволит не тратить лишние силы на разработку детальных моделей для уже имеющихся в наличии компонентов системы, которые доступны для использования на стенде или в виртуальной среде, что сокращает общее время и стоимость подготовки к проверке модели.

В направлении *динамического статического анализа моделей* проведены лишь исследовательские работы, поэтому реализация и проведение экспериментов с этим методом анализа является еще одной задачей для будущего развития функциональности инструмента.

В контексте *автоматического синтеза моделей* перспективными направлениями развития являются поддержка новых видов ограничений на синтезируемую модель, исследование методов инкрементального синтеза архитектуры и автоматического обновления модели при изменении исходных требований с учетом ручных модификаций предыдущих синтезированных моделей, а также синтез отказоустойчивых архитектур, которая бы с учетом информации о степени критичности каждой функции обеспечивала бесперебойную работу всей системы при условии возможности отказов отдельных компонентов.

Еще одним направлением для развития инструментов MASIW является генерация документации с описанием архитектуры системы КБО, а также генерация шаблонов проектов и исходного кода функциональных приложений, которые бы уже включали типовые функции, такие как обработка сообщений структура которых уже описана в архитектурной модели.

9. Заключение

Сложность современных авиационных систем и высокие требования к их надёжности приводят к необходимости использования разделяемых ресурсов (архитектура ИМА). При создании систем ИМА разработчики (в частности, системные интеграторы) сталкиваются с рядом задач и проблем, с которыми они не сталкивались ранее. Для решения этих задач приходят на помощь различные средства автоматизации и компьютерной поддержки разработки. Развитие этого направления в первую очередь связано с использованием разнообразных моделей, в том числе, архитектурных моделей программно-аппаратных систем. Соответствующая группа технологий получила название

разработки систем на основе моделей (Model Driven System Engineering - MDSE).

Внедрение в практику технологий MDSE требует серьезных исследований и продуманных инженерных решений. Одним из источников сложности развития и внедрения MDSE является необходимость учета потребностей и предпочтений различных групп специалистов, поскольку модели одновременно используются как исходные данные для синтеза и для верификации, как инструмент дизайна и как средство коммуникации и кооперации большого числа участников разработки. Методам и инструментам решения этих задач собственно и посвящена данная статья. Особое внимание в статье уделяется вопросам интеграции методов формальной спецификации и формального анализа моделей авионики с методами проектирования, реализации и интеграции систем авионики, которые были наработаны в этой отрасли ранее.

Инструмент MASIW, разрабатываемый в ИСП РАН в сотрудничестве с ГосНИИАС, упрощает решение ряда задач, связанных с разработкой авиационных систем. Он позволяет удобно и наглядно создавать и редактировать модели таких систем на языке AADL, а также проводить **анализ** таких моделей на соответствие разнообразным требованиям, связанным как со структурой, так и с поведением модели (вычислять разнообразные временны́е характеристики, прогнозировать поведение моделируемой системы в различных ситуациях, в том числе при нестандартном поведении компонентов и при отказах внутри системы).

Помимо этого, MASIW облегчает проектирование архитектуры за счет реализации ряда алгоритмов **синтеза** моделей. Это позволяет, в частности, распределить задачи по вычислительным блокам так, чтобы каждой задаче было выделено достаточно процессорного времени, и сгенерировать модель бортовой сети и схему распределения ресурсов сети в соответствии с потребностями компонентов системы.

Инструмент MASIW постоянно развивается. Это развитие опирается на тесную кооперацию с заказчиками, потенциальными пользователями и с международным сообществом разработчиков открытых стандартов и открытых инструментов для поддержки разработки, интеграции и верификации ответственных систем на основе использования средств моделирования.

Список литературы

[1] Е.А. Федосов. Авиация в России конца XX - начала XXI веков в статьях и интервью академика, М.: НИЦ ГосНИИАС, 2013.

[2] Gilbert Edelin. Embedded Systems at Thales: the Artemis challenges from an industrial perspective. ARTIST Summer School, 2009.

[3] Aircraft Data Network, Part 1: Systems Concepts and Overview, 2002.

[4] Aircraft Data Network, Part 2: Ethernet Physical and Data Link Layer Specification, 2002.

[5] Aircraft Data Network, Part 7: Deterministic Networks, 2003.
[6] Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling.
[7] Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit.
[8] Brett Murphy, Amory Wakefield. Early verification and validation using model-based design. The MathWorks, 2009.
[9] SAE International. “Architecture Analysis & Design Language (AADL)”, SAE International Standards document AS5506B, Nov 2004, Revised Mar 2012.
[10] The ATESST Consortium. East-adl 2.0 specification (November 2010), http://www.atesst.org
[11] ISO/IEC 19505-1:2012 Object Management Group Unified Modeling Language (OMG UML)
[12] Steve Vestal. MetaH Avionics Architecture Description Language. Honeywell Labs, 2001.
[13] Object Management Group (OMG). “Systems Modeling Language SysML”, Version 1.3.
[14] Object Management Group (OMG). “UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems.” Version 1.1.
[15] Dionisio de Niz, Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, SEI, 2007.
[16] O. Gilles, J. Hugues. Expressing and Enforcing User-Defined Constraints of AADL Models, Engineering of Complex Computer Systems (ICECCS), 2010.
[17] https://wiki.sei.cmu.edu/aadl/index.php/Osate_2_Lute
[18] S. Martin and P. Minet. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. Parallel and Distributed Processing Symposium, 2006.
[19] Jean-Yves Le Boudec, Patrick Thiran. Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Lecture Notes in Computer Science, vol. 2050, 2001.
[20] Jerry Banks, John S. Carson II. Discrete-Event System Simulation. Englewood Cliffs: Prentice-Hall, Englewood Cliffs: Prentice-Hall, 1984.
[21] S. Robinson, Discrete-Event Simulation: From the Pioneers to the Present, What Next? Journal of the Operational Research Society, 2005
[22] R.J. Ord-Smith, J.Stephenson, Computer, Simulation of Continuous Systems. Cambridge, 1975.
[23] Haynes, C. T., Friedman, D. P., and Wand, M., Continuations and coroutines. In Proceedings of the 1984 ACM Symposium on LISP and Functional Programming, LFP '84. ACM, New York, 1984.
[24] http://www.matthiasmann.de/content/view/24/26/
[25] AS5506/2 SAE Architecture Analysis and Design Language (AADL) Annex Volume 2.
[26] Brian R. Larson, Patrice Chalin, John Hatcliff, BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software, 10.1007/978-3-642-38088-4_19.
[27] L . Lamport. Proving the Correctness of Multiprocess Programs, 1977.
[28] С. Зеленов, Планирование строго периодических задач в системах реального времени, Труды Института системного программирования РАН, т. 20, 2011.
[29] Abdullah Al-Nayeem, Lui Sha, Darren D. Cofer, Steven M. Miller. Pattern-Based Composition and Analysis of Virtually Synchronized Real-Time Distributed Systems.

Proceedings of the Third International Conference on Cyber-Physical Systems – April 2012.
[30] P. Dissaux, F. Singhoff. Stood and Cheddar: AADL as a Pivot Language for Analysing Performances of Real Time Architectures, Proceedings of 4th International Congress ERTS-2008.
[31] F.Cadoret, E.Borde, S.Gardoll and L.Pautet. Design Patterns for Rule-based Refinement of Safety Critical Embedded Systems Models. International Conference on Engineering of Complex Computer Systems (ICECCS'12), Paris (FRANCE), 2012.

Tools for System Design of Integrated Modular Avionics

Buzdalov D.V., Zelenov S.V., Kornykhin E.V., Petrenko A.K., Strakh A.V., Ugnenko A.A., Khoroshilov A.V.
{buzdalov, zelenov, kornevgen, petrenko, strakh, ugnenko, khoroshilov} @ispras.ru

Abstract. Growth of modern avionics systems makes design of such systems impossible without involvement of automation. Nowadays an area of such tools is represented by both proprietary tools developed by the major aircraft manufacturers like Boeing and Airbus, and a number of open or partially open international projects varying in maturity, availability of source code and documentation. All the tools are based on architecture models of a system under design.

The paper considers languages available to describe architecture models of avionics systems and shows that AADL is the most appropriate one because of availability of textual notation and build-in concepts well suited to represent most elements of embedded systems.

Then the paper presents a toolset for design of modern avionics systems developed by ISPRAS in collaboration with GosNIIAS. The toolset named MASIW provides both a generic platform for design and analysis of architecture models and a specialized solution for the particular domain of avionics systems. It supports creation, editing and management of AADL models in both textual and graphical notation. Also MASIW provides various features for analysis and synthesis of AADL models.

Analysis capabilities include

- a checker of static structural constraints such as resource sufficiency, interface consistency, usage domain rules, etc.
- specialized analyzer of AFDX networks aimed to statically estimate latencies, buffer usage, etc.
- a simulator of AADL models augmented with behaviour specification in AADL Behaviour Annex notation or in Java.

Synthesis capabilities include schedule generation for a particular processor module as well as automatic building of assignment of hardware platform resources to software components in accordance with all requirements formalized in the architecture model.

Finally, MASIW provides a framework for generation of configuration data from architecture models. Currently it is used for generation of configuration tables for ARINC-653 operating systems and for AFDX switches and endpoints.

Keywords: Integrated Modular Avionics; IMA; Early Validation; model of platform; AADL.

References

[1]. E.A. Fedosov. Aviacija v Rossii konca XX - nachala XXI vekov v stat'jah i interv'ju akademika [Aviation in Russia in the late XX - early XXI centuries in articles and interviews of the Academician], M.: NIC GosNIIAS, 2013.
[2]. Gilbert Edelin. Embedded Systems at Thales: the Artemis challenges from an industrial perspective. ARTIST Summer School, 2009.
[3]. Aircraft Data Network, Part 1: Systems Concepts and Overview, 2002.
[4]. Aircraft Data Network, Part 2: Ethernet Physical and Data Link Layer Specification, 2002.
[5]. Aircraft Data Network, Part 7: Deterministic Networks, 2003.
[6]. Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling.
[7]. Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit.
[8]. Brett Murphy, Amory Wakefield. Early verification and validation using model-based design. The MathWorks, 2009.
[9]. SAE International. "Architecture Analysis & Design Language (AADL)", SAE International Standards document AS5506B, Nov 2004, Revised Mar 2012.
[10]. The ATESST Consortium. East-adl 2.0 specification (November 2010), http://www.atesst.org
[11]. ISO/IEC 19505-1:2012 Object Management Group Unified Modeling Language (OMG UML)
[12]. Steve Vestal. MetaH Avionics Architecture Description Language. Honeywell Labs, 2001.
[13]. Object Management Group (OMG). "Systems Modeling Language SysML", Version 1.3.
[14]. Object Management Group (OMG). "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems." Version 1.1.
[15]. Dionisio de Niz, Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, SEI, 2007.
[16]. O. Gilles, J. Hugues. Expressing and Enforcing User-Defined Constraints of AADL Models, Engineering of Complex Computer Systems (ICECCS), 2010.
[17]. https://wiki.sei.cmu.edu/aadl/index.php/Osate_2_Lute
[18]. S. Martin and P. Minet. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. Parallel and Distributed Processing Symposium, 2006.
[19]. Jean-Yves Le Boudec, Patrick Thiran. Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Lecture Notes in Computer Science, vol. 2050, 2001.
[20]. Jerry Banks, John S. Carson II. Discrete-Event System Simulation. Englewood Cliffs: Prentice-Hall, Englewood Cliffs: Prentice-Hall, 1984.
[21]. S. Robinson, Discrete-Event Simulation: From the Pioneers to the Present, What Next? Journal of the Operational Research Society, 2005

[22]. R.J. Ord-Smith, J.Stephenson, Computer, Simulation of Continuous Systems. Cambridge, 1975.
[23]. Haynes, C. T., Friedman, D. P., and Wand, M., Continuations and coroutines. In Proceedings of the 1984 ACM Symposium on LISP and Functional Programming, LFP '84. ACM, New York, 1984.
[24]. http://www.matthiasmann.de/content/view/24/26/
[25]. AS5506/2 SAE Architecture Analysis and Design Language (AADL) Annex Volume 2.
[26]. Brian R. Larson, Patrice Chalin, John Hatcliff, BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software, 10.1007/978-3-642-38088-4_19.
[27]. L. Lamport. Proving the Correctness of Multiprocess Programs, 1977.
[28]. S. Zelenov. Planirovanie strogo periodicheskih zadach v sistemah real'nogo vremeni [Scheduling of Strictly Periodic Tasks in Real-Time Systems]. Trudy ISP RAN [The Proceedings of ISP RAS], t. 20, 2011.
[29]. Abdullah Al-Nayeem, Lui Sha, Darren D. Cofer, Steven M. Miller. Pattern-Based Composition and Analysis of Virtually Synchronized Real-Time Distributed Systems. Proceedings of the Third International Conference on Cyber-Physical Systems – April 2012.
[30]. P. Dissaux, F. Singhoff. Stood and Cheddar: AADL as a Pivot Language for Analysing Performances of Real Time Architectures, Proceedings of 4th International Congress ERTS-2008.
[31]. F.Cadoret, E.Borde, S.Gardoll and L.Pautet. Design Patterns for Rule-based Refinement of Safety Critical Embedded Systems Models. International Conference on Engineering of Complex Computer Systems (ICECCS'12), Paris (FRANCE), 2012.