Научная статья
УДК 004.942
DOI: 10.31799/2949-0693-2023-2-52-63

Разработка методов получения прикладной имитационной модели бортовой сети на основе ее формального описания

Николай Иванович Синёв[1]
nikolay.sinyov@guap.ru, orcid.org/0000-0001-8421-4791
Валентин Леонидович Оленев[1]
✉ valentin.olenev@guap.ru, orcid.org/0000-0002-1817-2754
[1] Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, Российская Федерация

Аннотация. *Рассматриваются существующие подходы к разработке, тестированию и верификации современных бортовых протоколов и показан прототип программного обеспечения, которое на основе формальной модели сети создает прикладную модель. На основе анализа существующих решений сформированы требования к программному обеспечению, которое должно осуществлять данный переход. Сформулированы проблемы, которые планируется решить в ходе дальнейших исследований.*

Ключевые слова: *бортовые сети, коммуникационные протоколы, SystemC, сеть Петри, имитационное моделирование*

Для цитирования: *Синёв Н. И., Оленев В. Л. Разработка методов получения прикладной имитационной модели бортовой сети на основе ее формального описания // Инновационное приборостроение. 2023. Т. 2, № 2. С. 52–63. DOI: 10.31799/2949-0693-2023-2-52-63.*

Original article

Development of methods for obtaining an applied simulation model of the onboard network based on its formal description

Nikolay I. Sinyov[1]
nikolay.sinyov@guap.ru, orcid.org/0000-0001-8421-4791
Valentin L. Olenev[1]
✉ valentin.olenev@guap.ru, orcid.org/0000-0002-1817-2754
[1] Saint Petersburg State University of Aerospace Instrumentation, Saint Petersburg, Russia

Abstract. *The development, testing and verification of modern on-board protocols is a very knowledge-intensive task. It is necessary to develop a different range of models for each stage of new standard development to decide finally to verify and validate the new protocol. All this requires a lot of time. To reduce the development time, it is necessary to optimize the process of preparing the models. To do this it is necessary to transform the formal model of the new standard into an applied one. In this article the existing approaches will be considered and the prototype of software, which on the basis of formal network model creates an applied network model, will be shown.*

Keywords*: on-board networks, communication protocols, SystemC, Petri net, simulation modeling*

For citation: *Sinyov N. I., Olenev V. L. Development of methods for obtaining an applied simulation model of the onboard network based on its formal description. Innovacionnoe priborostroenie = Innovative Instrumentation. 2023;2(2):52–63. (In Russ.). DOI: 10.31799/2949-0693-2023-2-52-63.*

Введение

Разработка, тестирование и верификация современных бортовых протоколов является очень наукоемкой задачей. Кроме времени, которое затрачивается на разработку нового стандарта, также много сил уходит на валидацию и верификацию. Для решения этих задач разработчикам необходимо спроектировать формальные (сети Петри [1]), имитационные (программа с применением SystemC [2]) и физические (железные устройства) модели для создаваемого стандарта.

Разработка каждой такой модели требует подготовленной команды людей, которые имеют определенные компетенции в каждой из областей. На каждую из этих моделей затрачивается большое количество человеко-часов, которые занимают львиную долю всего времени разработки. Разработчики всегда хотят оптимизировать свою нагрузку и как можно скорее закончить проект. А что если можно оптимизировать две из трех

моделей? Этого можно добиться путем трансформации построенной формальной модели в прикладную через специальную программу-конвертор. Данный подход в два раза сократил бы время разработки и тестирования формальных и прикладных (имитационных) моделей и уменьшил затраты на разработку.

В данной статье рассматриваются существующие решения, подходы и первые наброски программы, которая на основе формальной модели сети Петри создает прикладную модель сети с использованием SystemC.

Существующие решения

MAS-CommonKADS

MAS-CommonKADS – это методология разработки агентно-ориентированного программного обеспечения или модели (агентно-ориентированная система – АОС), предназначенная для применения на этапах анализа и проектирования. Она является гибридом методологии инженерии знаний CommonKADS [3], объектно-ориентированной методологии «Техника объектного моделирования» (Object Modelling Technique – OMT), объектно-ориентированной инженерии программного обеспечения (Object-oriented Software Engineering – OOSE) и метода проектирования, получившего название «Проектирование, управляемое ответственностью» (Responsibility Driven Design – RDD). В жизненном цикле разработки системы в рамках данной методологии выделяется ряд этапов:

– *этап концептуализации* предполагает получение первичного описания решаемой проблемы посредством создания набора диаграмм использования. Результат может быть представлен и в виде текстового описания системы;

– на *этапе анализа* определяются функциональные требования к системе. Система описывается через определенный набор моделей;

– на *этапе проектирования* могут использоваться как нисходящий, так и восходящий подходы, проводится определение повторно используемых компонентов и компонентов, которые следует разработать в зависимости от целевой платформы. На входе этого этапа применяются модели, полученные на этапе анализа, а на выходе – набор спецификаций, готовый к реализации;

– на *этапе разработки и тестирования* производится кодирование и тестирование агентов, модели которых получены на ранних стадиях. Внедрение и эксплуатация системы происходят на этапе использования.

В рамках методологии определены следующие модели (рис. 1).

Агентная модель описывает возможности агентов к рассуждению, наличие сенсоров и эффекторов, предоставляемые сервисы, объединение агентов в группы и иерархии.

Модель задач формализует цели существования, декомпозицию целей, методы решения проблем.

- *Рис. 1. Взаимодействие моделей в методологии MAS-CommonKADS*
- *Fig. 1. Interaction of models in the MAS-CommonKADS methodology*

Экспертная модель определяет, какие знания необходимы агентам для достижения поставленных целей.

Организационная модель описывает организацию, в которой предстоит работать АОС, а также социальную организацию агентного сообщества.

Координационная модель описывает переговоры агентов при их взаимодействии и используемые протоколы.

Коммуникационная модель описывает взаимодействие человека с разрабатываемым программным обеспечением, а также факторы, влияющие на проектирование пользовательского интерфейса.

Проектная модель объединяет все предыдущие модели и состоит из трех подмоделей. Первая подмодель – это проект сети, где происходит проектирование соответствующих аспектов агентной инфраструктуры (необходимая сеть, знания и средства обработки и передачи данных). Вторая подмодель представляет агентное проектирование, служащее для разделения или объединения агентов, полученных на ранних этапах на основании соображений целесообразности и выбора подходящей архитектуры для каждого агента. Третья подмодель – платформозависимое проектирование, где происходит определение платформы разработки агентов для каждой из выбранных архитектур.

Основным отличием этой методологии от других является то, что разрабатываемая АОС рассматривается с различных точек зрения. Разработчик получает описание, позволяющее выполнять реализацию на различных программных платформах. Сильной стороной методологии принято считать использование стандартных методов инженерии программного обеспечения, которые расширяются естественным способом. MAS-CommonKADS создавалась в расчете на многоразовое использование материалов, полученных на каждом уровне проектирования, что дает возможность использования данных из других проектов. Главный недостаток этой методологии – слабая поддержка этапов проектирования, тестирования и кодирования.

Прикладные МАС

В работах В. А. Виттиха [4], П. О. Скобелева и С. В. Батищева предложена методика создания прикладных открытых МАС (MAS, англ. Multi-agent system – многоагентная система) на основе концепции сетей потребностей и возможностей (ПВ-сетей). Данная методика поддержана развитыми инструментальными средствами и обеспечивает эффективную реализацию прикладных МАС в рамках заявленной концепции. Однако она ориентирована на сравнительно узкий класс систем промышленной логистики и агентную модель, описывающую конкуренцию за ресурсы.

В качестве наглядного примера того, как теоретические модели МАС могут быть доведены до практического использования, можно привести деятельность Института проблем управления сложными системами РАН и НПК «Генезис знаний» (г. Самара), которые являются лидерами практического использования АОС в области государственного управления и в социальной сфере.

Coq

Coq – инструмент интерактивного доказательства теорем [5], позволяющий формализовать математические концепции, а затем помогающий в интерактивном режиме генерировать проверенные на машине доказательства теорем. Машинная проверка дает пользователям гораздо больше уверенности в правильности доказательств по сравнению с доказательствами, созданными и проверенными людьми. Coq использовался в ряде флагманских проектов верификации, в том числе в компиляторе C, верифицированном CompCert, и служил для проверки доказательства теоремы о четырех цветах (среди многих других математических формализаций).

Пользователи генерируют доказательства, вводя серию тактик, составляющих этапы доказательства. Существует множество встроенных тактик, некоторые из которых являются элементарными, а другие реализуют сложные процедуры принятия решений (например lia, процедура принятия решений для линейной целочисленной арифметики). Еще одна встроенная тактика – Ltac и его запланированная замена, Ltac2, предоставляют языки для определения новых тактик, комбинируя существующие тактики с циклами и условными конструкциями. Они позволяют автоматизировать большие части доказательств, а иногда и целые доказательства. Кроме того, пользователи могут добавлять новые тактики или функциональные возможности, создавая плагины Coq с помощью OCaml.

Ядро Coq [6] выполняет окончательную проверку правильности доказательства, сгенерированного тактикой. Обычно это доказательство действительно правильное, но делегирование проверки доказательства ядру означает, что даже если тактика ошибочна, она не сможет ввести в систему неверное доказательство.

Наконец, Coq также поддерживает извлечение проверенных программ в такие языки программирования, как OCaml и Haskell. Это позволяет эффективно выполнять код Coq и может исполь-

зоваться для создания проверенных программных библиотек.

SpGraph and Spark

Spark [7] – это унифицированный аналитический движок для обработки больших данных. Для него разработан GraphX – модуль обработки графов Spark, который служит для обработки графических данных, например, графа социальной сети, состоящего из более чем одного миллиарда людей и сотни миллиардов дружеских связей. GraphX, если сравнивать с другими системами (Pregel, GraphLab и Gemini), является наиболее популярным благодаря следующим возможностям:

– он совместим и широко используется с системами MapReduce/Hadoop [7];

– приложения, разработанные на основе Spark, работают годами. GraphX позволяет без нарушения работы этих приложений обрабатывать огромные массивы данных в режиме реального времени.

Lean

Lean – это средство для доказательства теорем и язык программирования [8], основанный на исчислении конструкций с индуктивными типами. Это проект с открытым исходным кодом, размещенный на GitHub. Он был запущен Леонардо де Моура из *Microsoft Research* в 2013 г.. Lean имеет интерфейс, который отличает его от других интерактивных средств доказательства теорем, может быть скомпилирован в JavaScript и доступен в веб-браузере. Он имеет встроенную поддержку символов Unicode (их можно набирать с помощью LaTeX-подобных последовательностей, таких как «times» для «×»). Lean также имеет обширную поддержку метапрограммирования.

Функции, которые выполняет язык Lean:

– вывод типа;
– первоклассные функции;
– мощные типы данных;
– сопоставление с образцом;
– типовые классы;
– расширяемый синтаксис;
– макросы по визуальной обработке кода;
– зависимые типы;
– фреймворк метапрограммирования;
– многопоточность.
– самоверификация: возможность доказать собственные функции на языке Lean.

Сравнение подходов

Для реализации собственного решения при проектировании бортовых сетей необходимо сравнить и проанализировать существующие на рынке варианты с целью выбора наиболее подходящего подхода, который можно будет применить для разрабатываемого программного обеспечения.

Многоагентные системы могут быть использованы для решения таких проблем, которые сложно или невозможно решить с помощью одного агента или монолитной системы. Примерами таких задач являются онлайн-торговля, ликвидация чрезвычайных ситуаций и моделирование социальных структур. Многоагентные системы обучаются на основе моделей мира, поведения человека, биржевых операций, работы физики и многого другого. Переход от формальных моделей к физическим является основной идеей машинного обучения.

В MAS-CommonKADS применяются модели, полученные на этапе анализа, а на выходе – получаем набор спецификаций, готовых к реализации. Данный идейный подход хорошо описывает цель работы, однако такие системы не были внедрены в разработку бортовых систем методом перехода от графов к программной реализации.

Прикладные МАС работают похожим образом и могут быть доведены до практического использования. МАС модели ориентированы на сравнительно узкий класс систем промышленной логистики и агентную модель, описывающую конкуренцию за ресурсы. Поэтому метод не полностью удовлетворяет целям данной работы.

Инструмент интерактивного доказательства теорем – программное обеспечение, помогающее исследователю в разработке формальных доказательств в процессе взаимодействия человека с машиной. Как правило, такое программное обеспечение включает в себя какую-то разновидность интерактивного редактора доказательств или другой интерфейс, с помощью которого человек может вести поиск доказательств, сведения о которых хранятся в компьютере, а также процедуры автоматической проверки доказательств, осуществляемые компьютером.

На основе языка Coq пользователь может формализовать математические методы, решения и доказательства, а затем проверять и модифицировать полученные от программы интерактивные представления. Таким образом, рассматриваемое ПО удовлетворяет задачам поставленной работы, но все же данный язык программирования работает лишь на математических представлениях, а не на структуре сетевых технологий.

Spark и SpGraph могут применяться для доказательства математических теорем. Они оперируют данными в виде графов, что сильно корре-

лирует с одной из целей работы – с сетью Петри, которая в свою очередь так же является графом.

Lean кроссплатформенен – работает на ОС Windows, Linux, IOS, Android. Существуют также вариации на Raspberry Pi. Помимо этого, он позволяет от формальной модели получить математические доказательства. Реализован на языке C++, так что наиболее вероятно может подойти для решения поставленной в статье задачи.

Проведя сравнение и рассмотрев методы перехода от формальных моделей к прикладным (табл. 1), можно сделать вывод о том, что ни один из подходов не предназначен для работы с сетями. Это значит, что напрямую использовать один из методов для разрабатываемой модели невозможно. Выделим частные характеристики рассмотренных подходов и составим требования для решения поставленной задачи.

1. Поддержка C++. Проект разрабатывается с применением C++ фреймворка SystemC, поэтому использование данного языка необходимо для реализации прикладной модели.

2. Работа с графами. Сеть Петри является ориентированным мультиграфом, поэтому необходимо использовать методы, позволяющие работать в области математической абстракции.

3. Работа с сетевыми протоколами. Для реализации бортовой сети, которая передает данные между устройствами, необходимо использовать сетевые протоколы или протоколы бортовых сетей.

Разрабатываемая модель

На основе составленных требований был выделен следующий порядок для разработки:

1. Составление модели сети Петри для части одного из существующих транспортных протоколов.

2. Создание конфигурационного файла, который будет хранить все настройки и разметку для формальной модели.

3. Создание программы-транслятора, которая будет переводить объекты формальной модели в объекты прикладной модели (генерация C++ структур, описывающих работу сети).

Рассмотрим основные моменты, которые используются при разработке.

Правила построения сетей Петри

Первым этапом разработки является разработка модели посредством сети Петри. Сети Петри – это специальные типы моделей, используемые при анализе данных, моделировании, моделировании бизнес-процессов и других сценариях [1]. Этот тип математической конструкции может помочь в планировании рабочих процессов или представлении данных о сложных системах.

Сети Петри используют такие элементы, как позиции, переходы и направления для описания сложных процедур и моделирования работы системы. Фишки и системы маркировки могут отображать перемещение по этим системам. Большая часть модели может быть представлена в классической математической нотации или в виде конкретных рисунков, соответствующих синтаксису и составу модели сети Петри.

Для иллюстрации понятий теории сетей Петри самым удобным является графическое представление. Теоретико-графовым представлением сети Петри является двудольный ориентированный мультиграф, а структура сети Петри представляет собой совокупность позиций и переходов.

- *Таблица 1. Сравнение характеристик подходов*
- *Table 1. Comparison of the characteristics of approaches*

Сравнение характеристик	Рассматриваемые реализации				
	MAS-CommonKADS	прикладные МАС	Coq	SpGRAPH/Spark	Lean
Поддержка C++	–	–	+	–	+
Применение в сетевых технологиях	–	–	–	–	–
Описание формальных моделей	+	+	+	+	+
Описание прикладных моделей	+	+	+	+	+
Наличие перехода между моделями	+	+	+	+	+
Использование сетей Петри	–	–	–	–	–
Техническое применение	+	+	–	–	–
Простота реализации	–	–	+	+	+
Графическое представление моделей	–	–	+	+	+
Интегрированная среда	–	+	+	+	+

Граф сети Петри обладает двумя типами узлов, показанными на рис. 2.

Позиции и переходы соединяют ориентированные дуги (стрелки), при этом некоторые из них направлены от позиций к переходам, а другие – от переходов к позициям. Сеть Петри допускает существование кратных дуг от одной вершины графа к другой. Кратность дуги означает, что позиция и переход связаны не одной дугой, а несколькими. Например, кратность дуги, равная четырем, показывает, что между позицией и переходом нарисованы четыре дуги, направленные в одну сторону.

Каждая дуга должна соединять элементы, принадлежащие разным множествам элементов – позиций (S) и переходов (T). Таким образом, в сети Петри дуг между двумя переходами или двумя позициями быть не может.

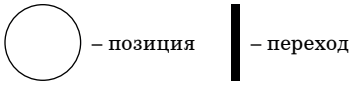
Маркировка сети Петри – это присвоение фишек ее позициям. Фишка (или маркер) – это примитивное понятие сетей Петри подобно позициям и переходам. Она может символизировать любой объект, движущийся по сети Петри. Фишки могут быть присвоены только позициям. Число и положение фишек при выполнении сети Петри могут изменяться. Это зависит от ее структуры.

Простое представление системы сетью Петри основано на двух основополагающих понятиях: событиях и условиях.

События – это действия, происходящие в системе. Возникновением событий управляет состояние системы. Состояние системы может быть описано множеством условий.

Условие есть предикат или логическое описание состояния системы. Оно может принимать либо значение «истина», либо значение «ложь». События являются действиями, а каждое событие в системе когда-либо происходит. Для того чтобы событие произошло, необходимо выполнение соответствующих условий

В сети Петри условия моделируются позициями, события – переходами. При этом входы перехода являются предусловиями соответствующего события, выходы – постусловиями. Возникновение события равносильно запуску соответствующего перехода. Выполнение условия представляется фишкой в позиции, соответствующей этому условию. Запуск перехода удаляет разрешающие фишки, представляющие выполнение предусловий, и образует новые фишки, которые представляют выполнение постусловий.



- *Рис. 2. Типы узлов сети Петри*
- *Fig. 2. Types of Petri Net nodes*

SystemC

SystemC удовлетворяет потребность в языке проектирования и проверки системы, охватывающем аппаратное и программное обеспечение. Это фреймворк, встроенный в стандартный C++ [9] путем расширения языка с использованием библиотек классов, который выражает архитектурные и другие атрибуты системного уровня в форме классов C++ с открытым исходным кодом. Он обеспечивает проектирование и проверку на системном уровне. Этот язык особенно подходит для моделирования разделения системы, для оценки и проверки назначения блоков аппаратным или программным реализациям, а также для проектирования и измерения взаимодействий между функциональными блоками. Ведущие компании в области интеллектуальной собственности (IP), автоматизации проектирования электроники (EDA), полупроводников, электронных систем и встроенного программного обеспечения в настоящее время используют SystemC для исследования архитектуры, предоставления высокопроизводительных аппаратных блоков на различных уровнях абстракции и разработки виртуальных платформ для совместного проектирования аппаратного и программного обеспечения. SystemC был стандартизирован Open SystemC Initiative (OSCI) и Accellera Systems Initiative и ратифицирован как IEEE Std. 1666™-2011.

Разработка программного обеспечения

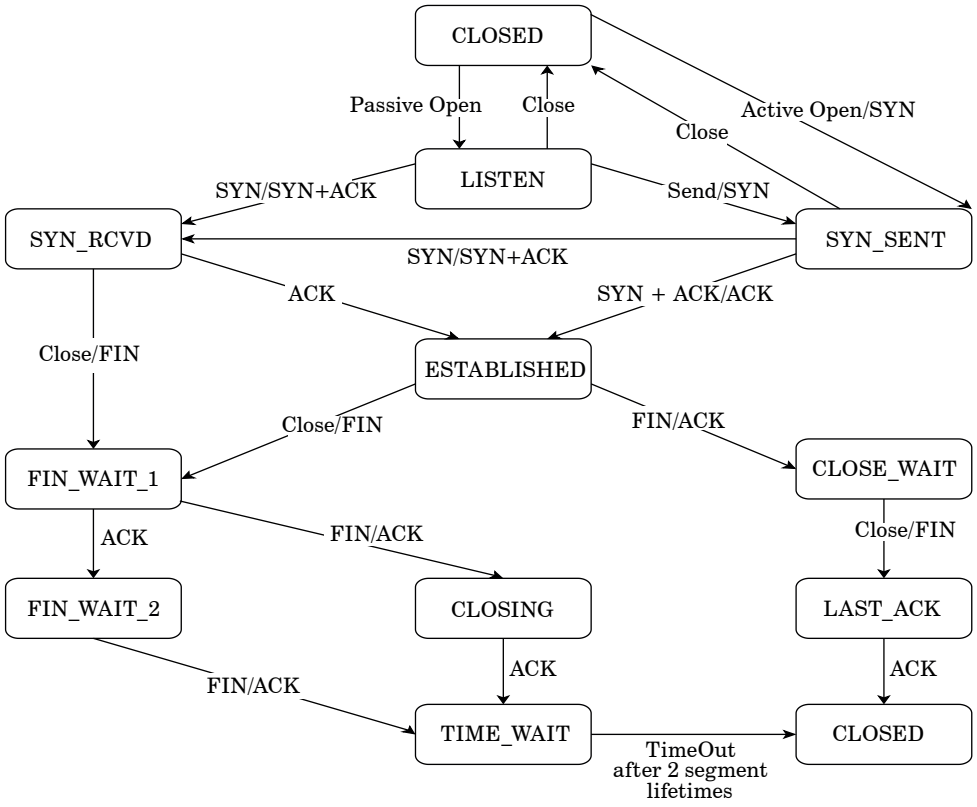
Реализация проекта начнется с построения сети Петри, удовлетворяющей правилам и теории инструментов моделирования. После чего необходимо разработать программу – симулятор движения фишек по сети с дальнейшим созданием конфигурационного файла. Далее по данным конфигурационного файла необходимо создать программное обеспечение, которое будет генерировать и изменять C++ структуры на основе теории сетей Петри и данных конфигурационного файла для дальнейшей работы с библиотекой SystemC. Ход разработки программного обеспечения представлен на рис. 3.

В качестве используемого протокола возьмем часть протокола транспортного уровня TCP/IP [10]. Возьмем схему состояний TCP (рис. 4) и нарисуем на ее основе сеть Петри (рис. 5).

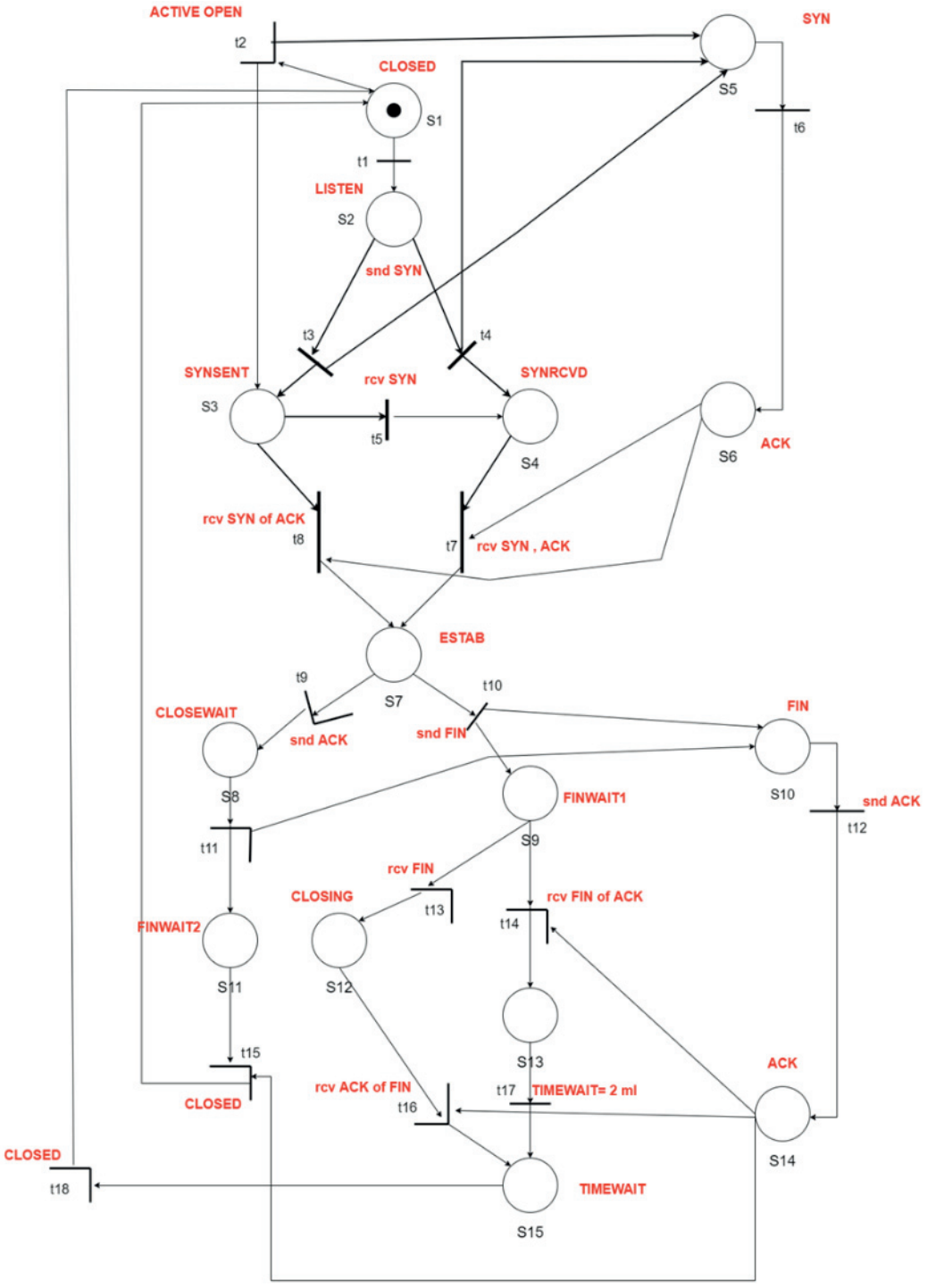
После построения сети Петри необходимо составить таблицу (табл. 2), где будет указано соответствие переходов и позиций.

Состояния, описанные в сети Петри, изменяются от команд прикладного уровня модели OSI,

- *Рис. 3. Схема разработки программного обеспечения*
- *Fig. 3. Software development scheme*



- *Рис. 4. Схема состояний TCP/IP*
- *Fig. 4. TCP/IP Status scheme*



- *Рис. 5. Спроектированная сеть Петри*
- *Fig. 5. Designed Petri Net*

таких как RECEIVE, SEND, CLOSE, OPEN, и, с другой стороны, под воздействием полей заголовка сетевого уровня.

В данной модели основными состояниями являются:

– CLOSED – режим автономной работы;

– LISTEN – режим прослушивания;

– ESTAB – режим установленного соединения.

Для того чтобы передать данные о сети Петри программе-симулятору, были составлены выходные множества для позиций O(s) и переходов O(t) (табл. 3).

На основе этих данных и начальной маркировки сети программа симулятор может произвести

- *Таблица 2. Значение позиций и переходов*
- *Table 2. Position and Transition values*

Позиция	Значение	Переход	Значение
s1	CLOSED	t1	AOPEN
s2	LISTEN	t2	POPEN
s3	SYNSENT	t3	SEND
s4	SYNRCVD	t4	rs
s5	SYN	t5	rs1
s6	ACK	t6	SENDACK
s7	ESTAB	t7	ras
s8	CLOSEWAIT	t8	rsa
s9	FINWAIT1	t9	rf
s10	FINWAIT1	t10	CLOSE2
s11	LASTACK	t11	CLOSE3
s12	CLOSING	t12	SENDACK
s13	FINWAIT2	t13	rf1
s14	ACK	t14	raf
s15	TIMEWAIT	t15	raf2
		t16	raf1
		t17	rf2
		t18	end

- *Таблица 3. Выходные множества O(s) и O(t)*
- *Table 3. Output sets O(s) and O(t)*

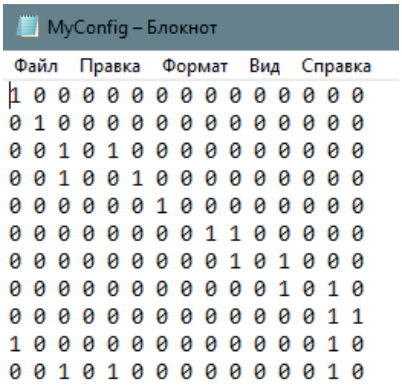
Выходные множества сети Петри	
множества для позиций	множества для переходов
O(s1) = {t1,t2}	O(t1) = {s2}
O(s2) = {t3,t4}	O(t2) = {s5,s3}
O(s3) = {t5,t8}	O(t3) = {s3,s5}
O(s4) = {t7}	O(t4) = {s4,s5}
O(s5) = {t6}	O(t5) = {s4}
O(s6) = {t7,t8}	O(t6) = {s6}
O(s7) = {t9,10}	O(t7) = {s7}
O(s8) = {t11}	O(t8) = {s7}
O(s9) = {t13,t14}	O(t9) = {s8}
O(s10) = {t12}	O(t10) = {s9,s10}
O(s11) = {t15}	O(t11) = {s11,s10}
O(s12) = {t16}	O(t12) = {s14}
O(s13) = {t17}	O(t13) = {s12}
O(s14) = {t14,t16,t15}	O(t14) = {s13}
O(s15) = {t18}	O(t15) = {s1}
	O(t16) = {s15}
	O(t17) = {s15}
	O(t18) = {s1}

```
Начальное состояние
1 0 0 0 0 0 0 0 0 0 0 0 0 0
Текущая маркировка: 1 0 0 0 0 0 0 0 0 0 0 0 0 0

Активные переходы:
1 2
Выберите активный переход: 1
Текущая маркировка: 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Активные переходы:
3 4
Выберите активный переход:
```

- *Рис. 6. Процесс моделирования формальной модели*
- *Fig. 6. The process of modeling a formal model*

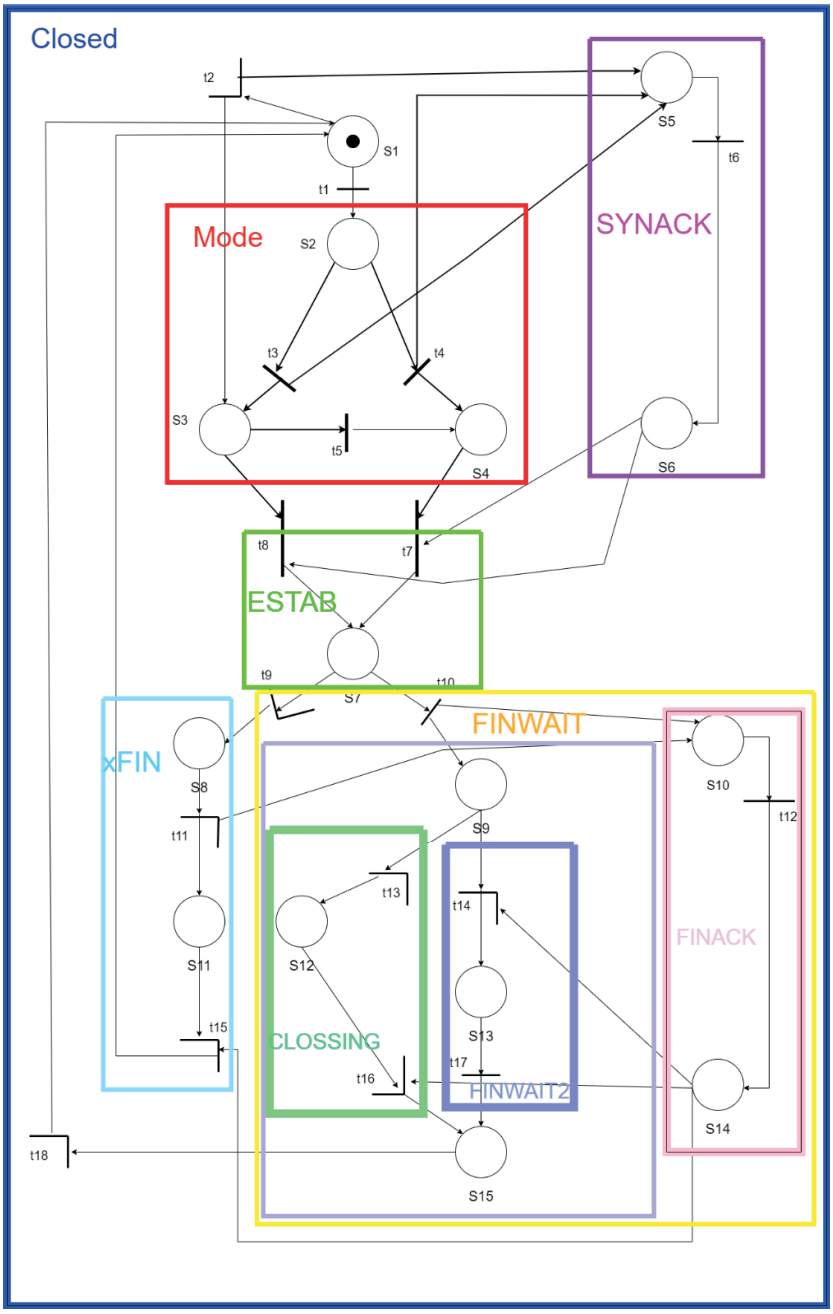
моделирование работы сети, которое показано на рис. 6.

По окончанию моделирования симулятор формальной модели готовит конфигурационный файл с матрицей (рис. 7) настроек для генерирования промежуточных текстов с кодом программы, который будет использоваться для генерации прикладной модели сети.



- *Рис. 7. Конфигурационный файл*
- *Fig. 7. Configuration file*

Для рассматриваемой части протокола TCP/IP было предложено разбиение на блоки формальной модели, чтобы потом каждый из полученных блоков генерировать в отдельную промежуточ-



- *Рис. 8. Разбиение модели на блоки*
- *Fig. 8. Splitting the model into blocks*

ную структуру. Данное разбиение представлено на рис. 8.

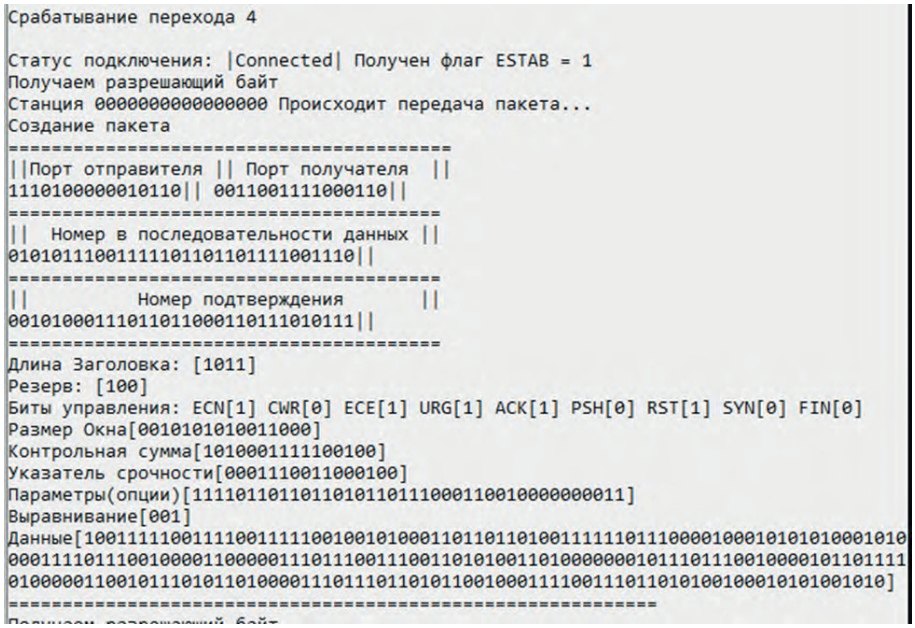
После этого на основе конфигурационного файла и преобразованных данных происходит компилирование и запуск имитационной модели на SystemC (рис. 9).

Полученная прикладная модель выполняет все возложенные на нее функции. Однако данный результат далек от финального решения:

– разбивка на блоки элементов формальной модели происходит вручную. Необходимо унифицировать структурное разделение и сделать процесс автоматизированным;

– формальная модель представлена только текстовым файлом. Нет возможности визуально посмотреть выполнение сети Петри. Необходимо доработать симулятор формальной модели;

– полученная прикладная имитационная модель использует малые возможности SystemC и по своей структуре представляет собой один большой файл, которые очень неудобен в обработке. Необходимо разбивать полученную модуль



- *Рис. 9. Процесс моделирования прикладной модели*
- *Fig. 9. The process of modeling an applied model*

на связные классы, которые должны быть соединены между собой на основе принципов объектно-ориентированного программирования.

Заключение

В данной статье был представлен первый прототип программного обеспечения, которое на основе формальной модели сети создает прикладную модель.

В статье были рассмотрены существующие подходы и решения в области перехода от формальной модели описания систем к их прикладным вариациям. На основе анализа существующих решений были сформированы требования к программному обеспечению, которое должно осуществлять переход от формальной модели сети к ее прикладной версии. Формальная модель такой сети должна описываться при помощи сетей Петри, а выходная прикладная модель строится с помощью языка программирования C++ и фреймворка SystemC.

Был построен прототип программного обеспечения, который выполняет базовые потребности по преобразованию описанной формальной модели в прикладную модель. Полученный прототип не лишен недостатков.

Усовершенствование программного обеспечения и устранение выявленных проблем планируется решить в ходе дальнейших исследований по тематике данной статьи.

СПИСОК ИСТОЧНИКОВ

1. *Оленев В. Л.* Моделирование систем: учеб. пособие. СПб.: ГУАП, 2015. 95 с.
2. SystemC. URL: https://systemc.org (дата обращения: 20.10.2022).
3. *Швецов А. Н.* Агентно-ориентированные системы: от формальных моделей к промышленным приложениям. Вологда: ВоГТУ, 2008. 101 с.
4. *Vittikh V. A., Skobelev P. O.* Multi-agent systems for modelling of self-organisation and co-operation processes // Proc. of XIII Intern. conference on the application of artificial intelligence in engineering. Ireland, Galway, 1998. P. 91–96.
5. QED at large: a survey of engineering of formally verified software / T. Ringer [et al.] // Foundations and trends® in programming languages. 2019. Vol. 5, № 2-3. 2019. P. 102–281.
6. *Трунов А.* Лекция 1. Введение в язык формальной верификации Coq // Лекториум. URL: https://www.youtube.com/watch?v=T_q5sQ5pQ7g (дата обращения: 20.10.2022).
7. PSGraph: How tencent trains extremely large-scale graphs with Spark? // IEEE 36th International conference on data engineering (ICDE). 2020. 1549–1557 P.

8. Lean language. URL: https://ru. wikipedia. org/wiki/Lean (дата обращения: 20.10.2022).
9. Open Standarts. URL: https://www.open-std. org/JTC1/SC22/WG21/ (дата обращения: 20.10.2022).
10. *Зайцев Д. А.* Верификация протокола TCP в процессе последовательной композиции модели Петри. Одесса: ОНАС, 2006. 15 с.

REFERENCES

1. Olenev V. L. Modeling of the systems: study guide. SPb.: SUAI; 2015. 95 p. (In Russ.).
2. SystemC. Available from: https://systemc.org [Accessed 20 October 2022].
3. Shvetsov A. N. Agent-based systems: from formal models to industrial applications. Vologda: VSTU; 2008. 101 p. (In Russ.).
4. Vittikh V. A., Skobelev P. O. Multi-agent systems for modelling of self-organisation and co-operation processes. Proc. of XIII Intern. conference on the application of artificial intelligence in engineering. Ireland, Galway; 1998, pp. 91–96.
5. Ringer T. et al. QED at large: a survey of engineering of formally verified software. Foundations and trends® in programming languages. 2019;5(2-3):102–281.
6. Trunov A. Lecture 1. Introduction to the language of formal verification Coq. Lectorium. Available from: https://www.youtube.com/watch?v=T_q5sQ5pQ7g [Accessed 20 October 2022].
7. PSGraph: How tencent trains extremely large-scale graphs with Spark?. IEEE 36th International conference on data engineering (ICDE). 2020, pp. 1549–1557.
8. Lean language. Available from: https://ru.wikipedia.org/wiki/Lean [Accessed 20 October 2022].
9. Open Standarts. Available from: https://www.open-std.org/JTC1/SC22/WG21/ [Accessed 20 October 2022].
10. Zaitsev D. A. Verification of the TCP protocol in the process of sequential composition of the Petri model. Odessa: ONAT; 2006. 15 p.

СВЕДЕНИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Синёв Николай Иванович, старший преподаватель кафедры аэрокосмических компьютерных и программных систем Санкт-Петербургского государственного университета аэрокосмического приборостроения.

Область научных интересов – имитационное моделирование, встроенные системы, бортовые вычислительные сети, коммуникационные протоколы.

Оленев Валентин Леонидович, кандидат технических наук, доцент, директор центра аэрокосмических исследований и разработок, заведующий кафедрой аэрокосмических компьютерных и программных систем Санкт-Петербургского государственного университета аэрокосмического приборостроения.

Область научных интересов – бортовые сети, встроенные системы, моделирование, формальная верификация, космические аппараты и техника, технология SpaceWire/SpaceFibre.

Sinyov Nikolay I., Senior Lecturer at the Department of Aerospace Computer and Software Systems, St. Petersburg State University of Aerospace Instrumentation.

Research interests – simulation modeling, embedded systems, on-board networks, communication protocols.

Olenev Valentin L., PhD in Technical Sciences, Associate Professor, Director of Aerospace R&D Centre, Head at the Department of Aerospace Computer and Software Systems, St. Petersburg State University of Aerospace Instrumentation.

Research interests – on-board networks, embedded systems, modeling, formal verification, spacecraft and equipment, SpaceWire/SpaceFibre technology.

Поступила в редакцию 26.10.2022
Поступила после рецензирования 14.11.2022
Принята к публикации 28.02.2023

Received 26.10.2022
Revised 14.11.2022
Accepted 28.02.2023