©Ицыксон В. М., 2016

DOI: 10.18255/1818-1015-2016-6-754-766

УДК 004.423.4+004.415.5

Формализм и языковые инструменты для описания семантики программных библиотек

Ицыксон В. М.

получена 5 сентября 2016

Аннотация. Статья посвящена вопросам спецификации структуры и поведения программных библиотек. Описываются существующие проблемы спецификации библиотек. Дается краткий обзор состояния дел в области формализации спецификации библиотек и библиотечных функций. Формулируются требования к создаваемому формализму. На основе требований предлагается формализм, позволяющий специфицировать все необходимые свойства библиотек, требуемые для автоматизации нескольких классов задач: обнаружение дефектов в программном обеспечении, миграция приложений в новое окружение, генерация программной документации. На базе формализма формулируются требования к языковым средствам спецификации библиотек. В заключении определяются дальнейшие направления исследований.

Ключевые слова: формальная спецификация, программная библиотека, поведенческое описание, программный дефект, язык спецификаций

Для цитирования: Ицыксон В. М., "Формализм и языковые инструменты для описания семантики программных библиотек", *Моделирование и анализ информационных систем*, **23:6** (2016), 754–766.

Об авторах:

Ицыксон Владимир Михайлович, orcid.org/0000-0003-0276-4517, канд. техн. наук, доцент, Санкт-Петербургский политехнический университет Петра Великого, ул. Политехническая, д. 29, г. Санкт-Петербург, 195251, Россия, e-mail: vlad@icc.spbstu.ru

1. Введение

Программные библиотеки – ставший стандартом «де-факто» способ реализации компонентно-ориентированного подхода, при котором производитель программного обеспечения инкапсулирует определенную функциональность в виде набора функций, типов данных и пользовательского программного интерфейса доступа. Современные библиотеки являются чрезвычайно сложными объектами, функциональность которых зачастую сложнее, чем у использующих их приложений.

Ключевым отличием библиотек от обычных приложений является способ их использования. Приложения применяют пользователи, следуя инструкциям, руководствам по эксплуатации и встроенным справкам, им не требуются формальные спецификации, описывающие приложения. В то же время основными пользователями

библиотек являются другие программисты, которым для интеграции функциональности приложений и библиотек требуется абсолютно четкое понимание, как работает библиотека, как ее можно использовать, как она влияет на приложение, какие изменения вносятся в нее от версии к версии и т.п.

Каким образом производитель библиотеки обычно специфицирует библиотеку? Обычно используется один или несколько следующих способов:

- заголовочные файлы с комментариями;
- словесное описание интерфейса библиотеки;
- словесное описание поведения отдельных функций;
- словесное описание некоторых допустимых последовательностей вызовов функций;
- поставляемые разработчиком примеры использования.

Однако ни один из указанных способов не решает задачи формальной спецификации семантики библиотеки. Семантика библиотеки состоит из двух составляющих: семантика отдельных функций и допустимых способов совместного использования разных функций библиотеки. Семантика отдельных функций определяется условиями вызова функции, получаемым результатом, побочными действиями и влиянием на окружение. Обычно семантика функций описывается неформально в виде текстовых описаний. Допустимые способы совместного использования разных функций библиотеки в лучшем случае описываются авторами неформально в сопроводительной документации к библиотеке, иногда с помощью сопутствующих примеров использования.

То есть, на настоящий момент в индустрии программной инженерии отсутствует аппарат формализованного описания семантики программных библиотек. В условиях отсутствия формальной спецификации библиотек существует несколько классов задач, которые невозможно удовлетворительно решить в настоящее время:

- автоматическая проверка корректности использования библиотеки каким-либо приложением. Здесь под корректностью мы будем понимать выполнение приложением протокола доступа к библиотеке, предусмотренного разработчиком;
- обнаружение программных ошибок в многофайловых проектах, использующих сторонние библиотеки в случае недоступности исходного кода;
- анализ совместимости приложения и новой версии библиотеки;
- портирование приложения в новое библиотечное окружение.

Таким образом, целью данной работы является создание формализма, позволяющего строго описывать все необходимые аспекты библиотек.

2. Обзор предметной области

Проблема спецификации библиотек и сервисов исследуется довольно давно. Имеется достаточное число публикаций, предлагающих разные подходы к описанию спецификаций. Первые исследования были связаны в первую очередь с обеспечением интероперабельности, а основная цель создаваемых спецификаций была создать самодостаточное описание интерфейсов библиотек и сервисов, для возможности использовать их в разных языках программирования и различных операционных системах. Примерами таких спецификаций является язык IDL [1], а также многие его расширения MIDL [2] или, например, ADL [3]. Основное ограничение этих языков для спецификации библиотек – ограниченность детальным описанием API, без фокусировки на валидных вариантах использования библиотек. То есть основной упор делается на описание сигнатур функций и типов данных, а спецификации семантики всей библиотеки не уделяется должного внимания.

Одним из первых исследований в области спецификации интерфейсов компонентов является работа Алена и Гарлана [4], в которой авторы сводят задачу взаимодействия компонентов программной системы к задаче спецификации протоколов взаимодействия наподобие протоколов компьютерных сетей. В качестве основы формализма был взят аппарат взаимодействующих последовательных процессов (CSP) Ч. Хоара [5], который они соответствующим образом видоизменили. Введя в формализм специальные элементы – порты, коннекторы и роли, они сделали возможным отдельно специфицировать разные аспекты возможного взаимодействия компонентов между собой. Использование формализма позволяет частично решать задачу совместимости компонентов с помощью верификатора FDR [6].

Альфарио и Хензингер в статье 7 предлагают свой формализм для описания взаимодействующих компонентов, названный интерфейсными автоматами. В работе используется оптимистическое определение совместимости компонентов, основанное на использовании модели окружения. Авторами предлагаются формальные методы проверки оптимистической совместимости двух интерфейсных автоматов.

Отдельные исследования посвящены не столько спецификациям интерфейсов и библиотек, сколько механизмам их автоматизированного построения на основе анализа существующего программного обеспечения. Так, в статье [8] авторами предлагается подход к выводу спецификаций библиотек на основе статического извлечения предикатов. Авторы используют анализ потоков данных (data flow analysis) и анализ потока управления для извлечения предикатов, характеризующих функции интерфейса. Другой подход изложен в работе [9], где авторы предлагают использование динамического вывода спецификаций библиотек на основе модульного тестирования. Для этого используется формализм Datalog, в терминах которого определяются функции библиотеки, интерфейсы, типы данных и транзакции. Валидные последовательности вызовов функций задаются с помощью специальных предикатов. Вывод спецификаций основывается на анализе и обобщении (генерализации) результатов случайного модульного тестирования функций библиотеки.

Один из наиболее интересных подходов к описанию API библиотек и правил их применения используется в подходе SLAM, предложенном Microsoft Research для верификации драйверов. SLAM использует язык SLIC [10] для спецификации библиотек и правил взаимодействия программ и API. SLIC-спецификация используется

для инструментирования программы и/или библиотеки для дальнейшего динамического или статического контроля соответствия. Отсутствие семантических описаний внутренностей библиотек не позволяет использовать его для автоматизации миграции.

Г. Ливинс в статье «The future of library specification» [11], кроме известных подходов, связанных с неформальным документированием и формальным специфицированием, выделяет несколько косвенных подходов: спецификация через примеры использования, спецификация посредством исходных кодов библиотек и спецификация через модульные тесты. Основной вывод автора – спецификация библиотек должна объединять все перечисленные подходы.

В наших предыдущих работах [12], [13] мы также предлагали формализм для спецификации библиотек и язык, поддерживающий задание таких спецификаций. Однако в этом подходе варианты использования библиотек (поведение) описываются неявно, а язык не позволяет в полной мере задавать контракты функций и влияние функций на окружение. Таким образом, на настоящий момент не существует универсального подхода к спецификации библиотек, позволяющего:

- детально описывать внешний интерфейс библиотеки;
- задавать возможные протоколы использования библиотеки;
- специфицировать побочные действия библиотеки – влияние ее на окружение;
- явно вводить семантические описания поведения библиотек.

3. Особенность организации библиотек

Особенность использования библиотек заключается в том, что библиотека не является чисто функциональным объектом, может иметь внутреннее состояние и различные побочные эффекты, существенно влияющие на возможность вызова отдельных функций.

Введем классификацию библиотек функций с точки зрения их внутреннего состояния.

1. Библиотеки без внутреннего состояния
2. Библиотеки с внутренним состоянием библиотеки
3. Библиотеки с внутренним состоянием создаваемого объекта
4. Комбинированные библиотеки

К первому классу относятся библиотеки, содержащие чистые функции без побочных эффектов. К таким библиотекам относится, например, библиотека математических функций `math.h` стандартной библиотеки языка программирования C.

Ко второму классу относятся библиотеки, которые хранят свое состояние. То есть поведение отдельных функций зависит от того, в каком состоянии находится библиотека. Примером такой библиотеки может служить, например, часть библиотеки `stdlib.h`, обеспечивающая генерацию случайных чисел. Вызов `srand()` задаёт

начальное значение генератора, а `rand()` выдает очередное случайное число из последовательности, построенной на основе начального значения генератора.

К третьему классу относятся библиотеки, которые сохраняют контекст внутри создаваемого функциями библиотеки объекта. Таким объектом может быть, например, созданный сокет или дескриптор файла. В первом случае контекст содержит параметры сокета (IP-адреса, номера портов, состояние), во втором случае контекст содержит параметры файла, режим открытия и указатель на следующие читаемые данные.

Четвертый класс – наиболее общий, к нему относятся библиотеки, объединяющие особенности второго и третьего класса.

4. Формальная спецификация библиотек

Полная формальная спецификация библиотек должна описывать:

- сигнатуру всех функций, входящих в библиотеку;
- контракт каждой библиотечной функции (предусловия, постусловия, влияние на окружение);
- поведенческую модель библиотеки, учитывающую все возможные варианты использования функций библиотеки, специфицирующую, в том числе, поведение библиотеки при не валидном ее использовании.

Исходя из вышеперечисленного предлагается полную спецификацию библиотек определять как $\langle F, L \rangle$, где

- $F = \{F_i\}$ – множество функций библиотеки;
- L – поведенческое описание библиотеки.

Отдельная функция библиотеки F_i определяется как

$\langle Name, Args, Res, Pre, Post, A, CondA, D, CondD, E, CondE \rangle$, где

- $Name$ – имя функции;
- $Args$ – множество формальных аргументов функции;
- Res – результат функции;
- Pre – предусловия функции, выраженные формулой в логике первого порядка от аргументов $Args$;
- $Post$ – постусловия функции, выраженные формулой в логике первого порядка от аргументов $Args$ и Res ;
- $A = \{A_i\}$ – множество семантических действий, производимых функцией;

- $CondA = \{CondA_i\}$ – множество условий выполнения семантических действий. Действие A_i выполняется во время работы функции тогда, когда выражение $CondA_i$ истинно.
- $D = \{D_i\}$ – множество запускаемых дочерних автоматов;
- $CondD = \{CondD_i\}$ – множество условий запуска дочерних автоматов. Автомат D_i запускается во время работы функции тогда, когда выражение $CondD_i$ истинно;
- $E = \{E_i\}$ – множество объектов окружения, изменяемых функцией (то есть объектов программы, использующей библиотеку);
- $CondE = \{CondE_i\}$ – множество условий изменения объектов окружения. Объект E_i изменяется во время работы функции тогда, когда выражение $CondE_i$ истинно.

Объединение всех множеств E объектов окружения, изменяемых всеми функциями, показывает множество всех объектов программы на которые влияет библиотека.

Поведенческое описание библиотеки будем представлять с помощью множества параметризуемых расширенных конечных автоматов (EFSM):

$$L = \{B, S_1(q, P), \dots, S_n(q, P), I\}, \text{ где}$$

- B – основной расширенный конечный автомат, описывающий поведение всей библиотеки;
- S_i – i -й дочерний расширенный конечный автомат, запускаемый при достижении определенных условий;
- q – параметр – начальное состояние дочернего автомата;
- P – дополнительный параметр дочернего автомата;
- I – множество созданных при инициализации библиотеки экземпляров дочерних автоматов (например, для библиотеки работы с файлами это могут быть автоматы, соответствующие файловым потокам `sysin`, `sysout` и `syserr`).

Состояние основного автомата соответствует состоянию библиотеки, состояния дочерних автоматов соответствуют состоянию созданных объектов. Стимулами автоматов, форсирующими переход автомата из одного состояния в другое, являются вызовы функций API библиотеки. Отдельный автомат определяется как модифицированный расширенный конечный автомат

$$\langle Q, Q_0, X, V, C, C^A, C^D, T \rangle, \text{ где}$$

- Q – множество управляющих состояний автомата – состояния библиотеки или объектов библиотеки;

- Q_0 – непустое множество начальных состояний автомата. Начальных состояний может быть несколько для дочерних автоматов, так как при создании экземпляра автомата начальные состояния могут быть разные, для автомата B начальное состояние всегда одно;
- X – множество состояний завершения. Дочерние автоматы после попадания в такие состояния уничтожаются;
- V – множество внутренних переменных автомата;
- $C = \{C_i\}$ – множество стимулов – вызовов функций, C_i – вызов i -й функции; $C_i \in F$;
- $C^A = \{C_i^A\}$ – множество семантических действий автомата, C_i^A – семантические действия, инициируемые запуском i -й функции при условии истинности C_i^{CondA} ;
- $C^D = \{C_i^D\}$ – множество дочерних автоматов, C_i^D – дочерние автоматы, запускаемые i -й функцией при условии C_i^{CondD} ;
- T – отношение перехода.

Из-за ограниченности объема статьи формализм представлен без глубокой детализации. За рамками описания остались такие вопросы, как спецификация невалидного поведения, действия по умолчанию, типы данных и т.п. Эти вопросы более подробно будут рассмотрены в других работах.

Вообще для разработчика поведенческое описание библиотек более наглядно представлять не в теоретико-множественном виде, а в графическом виде.

Пример описания в графическом виде клиентской части библиотеки TCP-сокетов представлен на рис. 1. На рисунке сплошной линией показаны переходы автоматов, пунктирной линией – запуски дочерних автоматов, состояния завершения выделены темно-серым цветом. Автомат **L** описывает общее поведение библиотеки **bsd-socket**, которая в отличие от **WinSock** не требует инициализации. Побочным результатом вызова **socket()** является создание нового автомата **P**, соответствующего созданному сокету, обладающего своим жизненным циклом. При этом стоит отметить, что создаваемых автоматов может быть несколько, отличаться они будут только параметром запуска дочернего автомата (элемент, соответствующий вызову **socket()**).

Более сложный пример представлен на рис. 2. Здесь приведена графическая модель серверной части библиотеки протокола TCP библиотеки **bsd-socket**. Кроме автомата верхнего уровня, соответствующего библиотеке (**L**), на рисунке представлены два семейства автоматов: одно (**P**) – инкапсулирует свойства слушающих сокетов, второе (**S**) – созданных серверных сокетов.

Оба примера демонстрируют только поведенческое описание библиотек, без спецификации множества функций.

Рис. 1. Пример простого автомата, соответствующего клиентской части протокола TCP библиотеки `bsd-socket`

Fig. 1. An example of a simple FSM representing the client side of the TCP protocol of the `bsd-socket` library

Рис. 2. Пример автомата, описывающего серверную часть протокола TCP библиотеки `bsd-socket`

Fig. 2. An example of a simple FSM representing the server side of the TCP protocol of the `bsd-socket` library

5. Перспективы использования разработанного формализма

5.1. Построение спецификаций

Предполагается два пути построения формальных спецификаций: с помощью авторов библиотек и сообщества разработчиков.

В первом случае спецификацию библиотеки создает ее разработчик. Для этого формируется язык описания спецификации библиотек (наподобие разработанного ранее языка PanLang [13]), средствами которого можно будет задать все свойства библиотеки, выражаемые разработанным формализмом. Требования к такому языку представлены в следующем разделе.

Во втором случае будут использованы методы вывода, базирующиеся на эксплуатации мирового программистского опыта (Empirical Software Engineering), когда структурная и часть поведенческой составляющих спецификаций строятся на основе анализа программных репозиториях (Mining Software Repositories). В этом случае формируется не вся спецификация, а только скелет, оставшуюся часть необходимо будет доработать вручную.

Методы анализа программных репозиториях для извлечения скелетов спецификаций в настоящее время разрабатываются в научной группе автора статьи, их описание выходит за рамки данной работы.

5.2. Использование формальных спецификаций библиотек

Разработанный и описанный в данной статье формализм может использоваться в дальнейшем для решения целого ряда научно-технических задач. К ним относятся: автоматизация поиска дефектов в сложных многокомпонентных программных проектах, автоматизация портирования приложений на новые библиотеки и автоматизация создания программной документации.

В рамках решения задачи поиска программных дефектов спецификации библиотек применяются для сокращения размерности задачи обнаружения. Это достигается путем аппроксимации поведения библиотек и библиотечных функций укрупненным видимым поведением, заданным в спецификации. В этом случае библиотечная функция заменяется системой предикатов, основанных на контрактах и на спецификации ошибочных состояний, заданных в спецификации. Такой подход используется в разрабатываемом в лаборатории анализа и верификации программ СПбПУ анализаторе Borealis [14], основанном на методе ограниченной проверки моделей (Bounded Model Checking, BMC). Похожий подход используется в основанном на абстрактной интерпретации инструменте Aegis, разрабатываемом в той же лаборатории [15].

Задача автоматизации миграции программ на новые библиотеки требует не только наличия внешней спецификации поведения библиотеки, но и частичного описания внутренней семантики библиотеки. На основе описания внутренней семантики строится семантический домен библиотеки, который может использоваться для проверки совместимости библиотеки и автоматического построения процедуры миграции [16].

6. Требования к языковым средствам задания спецификаций

Описанный в предыдущих разделах формализм удобен для доказательства свойств библиотек, а также для выполнения различных формальных процедур. Авторам библиотек и программистам нужны более удобные языковые инструменты, которые дадут возможность описывать поведение функций и библиотек в более привычном виде.

С этой целью в лаборатории верификации и анализа программ СПбПУ разрабатывается новое поколение языковых средств, полностью соответствующих разработанному формализму. Комплект языковых средств состоит из собственно языка

задания спецификаций и графического инструментария, позволяющего визуально описывать примитивы спецификаций – автоматы в виде графов переходов.

За основу для проектирования нового языка описания взят разработанный автором ранее язык описания библиотечных функций PanLang [13], используемый в статическом анализаторе Aegis [15]. При этом учитывается опыт других существующих языков, таких, например, как SLIC [10] и ACSL [17].

Основные функциональные требования к языку являются следствием представленного в данной работе формализма и его компонентов. Исходя из этого в язык должны входить следующие компоненты:

1. Описание основного расширенного конечного автомата, задающего поведение всей библиотеки (B).
2. Описание дочерних расширенных конечных автоматов, запускаемых при создании ресурсов библиотеки (S_i) с возможностью задания начального состояния q и параметра P .
3. Описания поведений функций (F_i), входящих в библиотеку.
4. Описание семантических доменов библиотеки. Каждый семантический домен вводит семантику решаемой высокоуровневой задачи. Такой задачей может быть проверка совместимости библиотек, миграция приложений, обнаружение дефектов или аппроксимация функций.
5. Описание семантических типов – специальных типов данных, унаследованных от классических типов данных, но имеющих проаннотированные имена и значения.
6. Описание процесса инициализации библиотеки, в том числе объектов и переменных, создаваемых на этапе инициализации.

Спецификация каждой функции должна содержать:

- описание сигнатуры функции, включающее, кроме названия функции, спецификацию аргументов ($Args$) и возвращаемого результата (Res), тип которых может быть в том числе и семантическим типом, обогащенным аннотациями;
- предусловия функции ($Pred$), заданные в виде предикатов от аргументов ($Args$), описывающие условия, при которых функция гарантирует корректность своего поведения (постусловий);
- постусловия функции ($Post$), заданные в виде предикатов от аргументов ($Args$) и возвращаемого результата (Res), описывающие гарантии поведения функции при условии выполнения предусловий;
- описания влияний функции на окружение. Каждое влияние описывается внешним изменяемым объектом (E_i) и предикатом ($CondE_i$), задающим условия, при котором объект изменяется;

- описания создаваемых при работе функции дочерних автоматов. Каждый создаваемый автомат характеризуется начальным состоянием и параметром. Создание дочернего автомата D_i происходит в зависимости от предиката создания $CondD_i$;
- описания влияний функции на окружение. Каждое влияние описывается внешним изменяемым объектом (E_i) и предикатом ($CondE_i$), задающим условия, при котором объект изменяется;
- описание действий (A). Каждое действие (A_i) является способом задания семантики поведения функции путем отображения на какой-либо семантический домен, зависящий от решаемой задачи. Таким доменом, например, может быть домен программных дефектов, детектируемых внутри библиотечной функции или библиотеки, или домен видимого поведения, требуемый для решения задачи проверки совместимости библиотек при миграции.

Одним из важных отличий создаваемого языка от языка Panlang является отсутствие задаваемого явно в императивном виде поведенческого описания функции (тела функции). Вместо этого все составляющие поведения (контракты, влияние на окружение, создание ресурсов, совершение семантических действий) задаются декларативно с помощью логических предикатов, представленных в виде формул логики первого порядка. Указанная особенность позволит, с одной стороны, обеспечить достаточную мощность описаний, а с другой – формально проверять различные свойства библиотек.

Нефункциональные требования к языку предъявляются следующие:

- простая грамматика и, как следствие, возможность построения эффективного парсера;
- прозрачный синтаксис и, следовательно, простота освоения пользователями.

На основании вышеперечисленных требований в настоящее время разрабатывается новый язык спецификации библиотек. Детальное описание синтаксиса и семантики разрабатываемого языка выходит за рамки настоящей статьи и является предметом отдельной публикации.

7. Заключение

В работе представлены результаты исследования по созданию формализма для спецификации программных библиотек. Формализм построен с учетом целого спектра задач, которые можно будет решать с его помощью. Основная идея – использовать одну и ту же формальную спецификацию как основу для нескольких методов программной инженерии: обнаружение программных дефектов, автоматизированная миграция программ и генерация программной документации. Из-за ограничений объема статьи формализм представлен без необходимой детализации. На основе разработанного формализма определены требования к языковым средствам спецификации библиотек.

Направление дальнейших исследований – разработка языковой поддержки для предложенного формализма и реализация конвертеров языковых описаний для существующих средств обнаружения ошибок и миграции программ.

Список литературы / References

- [1] Lamb D., “IDL: sharing intermediate representations”, *ACM Trans. Program. Lang. Syst.*, **9:3** (1987), 297–318.
- [2] Exton C., Watkins D., Thompson D., “Comparisons between CORBA IDL and COM/DCOM MIDL: Interfaces for Distributed Computing”, *Proceedings of the Technology of Object-Oriented Languages and Systems – Tools-25 (TOOLS '97)*, IEEE Computer Society, Washington, DC, USA, 1997, 15–23.
- [3] Sankar S., Hayes R., “ADL—an interface definition language for specifying and testing software”, *SIGPLAN*, **29:8** (1994), 13–21.
- [4] Allen R., Garlan D., “Formalizing architectural connection”, *Proceedings of the 16th international conference on Software engineering (ICSE '94)*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994, 71–80.
- [5] Хоар Ч., *Взаимодействующие последовательные процессы*, М.: Мир, 1989; [Hoar C.A.R., *Communicating sequential processes*, Mir, Moscow, 1989, (in Russian).]
- [6] Roscoe A.W., “Modelling and verifying key-exchange protocols using CSP and FDR”, *Proceedings of 1995 IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995.
- [7] de Alfaro L., Henzinger T., “Interface automata”, *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-9)*, ACM, New York, NY, USA, 2001, 109–120.
- [8] Ramanathan M., Grama A., Jagannathan S., “Static specification inference using predicate mining”, *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07)*, ACM, New York, USA, 2007, 123–134.
- [9] Sankaranarayanan S., Ivancic F., Gupta A., “Mining library specifications using inductive logic programming”, *Proceedings of the 30th international conference on Software engineering (ICSE '08)*, ACM, New York, USA, 2008, 131–140.
- [10] Ball T., Rajamani S.K., *SLIC: a Specication Language for Interface Checking (of C)*, Microsoft Research, Technical Report, MSR-TR-2001-21, 2002.
- [11] Leavens G.T., “The future of library specification”, *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10)*, ACM, New York, USA, 2010, 211–216.
- [12] Ицыксон В.М., Зозуля А.В., “Формализм для описания частичных спецификаций компонентов программного окружения”, *Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление*, **4** (2011), 81–90; [Itsykson V.M., Zozulya A.V., “The formalism for description of the partial specifications of program environment components”, *St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunication and Control Systems*, **4** (2011), 81–90, (in Russian).]
- [13] Ицыксон В.М., Глухих М.И., “Язык спецификаций поведения программных компонентов”, *Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление*, **3** (2010), 63–71; [Itsykson V.M., Glukhikh M.I., “A program component behavior specification language”, *St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunication and Control Systems*, **3** (2010), 63–71, (in Russian).]
- [14] Akhin M.Kh., Belyaev M.A., Itsykson V.M., “Software defect detection by combining bounded model checking and approximations of functions”, *Automatic Control and Computer Sciences*, **48:7** (2014), 389–397.

- [15] Itsykson V., etc, "Automatic defects detection in industrial C/C++ software", *Proceeding of 5th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR)*, IEEE, 2009, 50–55.
- [16] Ицыксон В.М., Зозуля А.В., "Автоматизированная трансформация программ при миграции на новые библиотеки", *Программная инженерия*, **6** (2012), 8–14; [Itsykson V.M., Zozulya A.V., "Automated Program Transformation for Migration to New Libraries", *Software Engineering*, **6** (2012), 8–14, (in Russian).]
- [17] Kirchner F., etc, "Frama-C: A software analysis perspective", *Formal Aspects of Computing*, **27**:3 (2015), 573–609.

Itsykson V.M., "The Formalism and Language Tools for Semantics Specification of Software Libraries", *Modeling and Analysis of Information Systems*, **23**:6 (2016), 754–766.

DOI: 10.18255/1818-1015-2016-6-754-766

Abstract. The paper is dedicated to the specification of the structure and the behaviour of software libraries. It describes the existing problems of libraries specifications. A brief overview of the research field concerned with formalizing the specification of libraries and library functions is presented. The requirements imposed on the formalism designed are established; the formalism based on these requirements allows specifying all the properties of the libraries needed for automation of several classes of problems: defects detection in the software, migration of applications into a new environment, generation of software documentation. The requirements on the language tools based on the developed formalism are proposed. The conclusion defines potential directions for further research.

Keywords: formal specification, software library, behavioral description, software defect, specification language

About the authors:

Vladimir M. Itsykson, orcid.org/0000-0003-0276-4517, PhD,
Peter the Great St. Petersburg Polytechnic University,
29 Polytechnicheskaya str., Saint-Petersburg 195251, Russia, e-mail: vlad@icc.spbstu.ru