УДК 004.4′2
doi:10.15217/issn1684-8853.2017.6.66

ОБЗОР СТАТИЧЕСКИХ МЕТОДОВ ВОССТАНОВЛЕНИЯ ЧАСТИЧНЫХ СПЕЦИФИКАЦИЙ ПРОГРАММНЫХ БИБЛИОТЕК НА ОСНОВЕ АНАЛИЗА ПРОГРАММНЫХ ПРОЕКТОВ

И. С. Егорова[a], *аспирант*
В. М. Ицыксон[a], *канд. техн. наук, доцент*
[a]*Санкт-Петербургский политехнический университет Петра Великого, Санкт-Петербург, РФ*

Постановка проблемы: *широкое использование сторонних библиотек и фреймворков в процессе разработки программного обеспечения и отсутствие исчерпывающей документации, специфицирующей их применение, делают актуальной задачу построения формальных спецификаций библиотек. Из-за отсутствия документации необходимы методы, позволяющие восстановить такие спецификации на основе множества успешных примеров применения библиотек в открытых проектах по разработке программного обеспечения.* ***Цель исследования:*** *анализ и классификация перспективных подходов к автоматизированному извлечению спецификаций программных библиотек, реализуемых на основе статических методов.* ***Результаты:*** *произведены обзор, классификация и сравнение различных формализмов, которые используются для описания библиотечных компонентов. Выявлено, что наиболее выразительные комплексные спецификации, позволяющие захватывать как временные свойства, так и ограничения на данные, используемые библиотечными компонентами, в большинстве подходов описываются с помощью расширенных конечных автоматов. Произведен обзор подходов, которые могут быть использованы для извлечения стандартных (частых) правил методами статического анализа; восстановлен обобщенный алгоритм, лежащий в их основе; осуществлено сравнение качества результирующих спецификаций. Выявлено, что большая часть методов, основанных на статическом анализе кода, предполагает использование шаблонов для задания множества искомых правил. Часто извлекаемые таким образом правила должны описывать жизненный цикл объектов единственного класса. Показано, что качество конечно-автоматных спецификаций, извлекаемых таким образом, является довольно низким. Продемонстрировано, что для получения более качественного результата необходимо использовать комплексные спецификации, методы извлечения которых малочисленны и на настоящий момент не систематизированы.*

Ключевые слова *— спецификация программной библиотеки, формальная спецификация, извлечение спецификаций, расширенные конечные автоматы, временные свойства, поведенческая модель библиотеки, комплексные правила, статический анализ.*

Введение

В настоящее время практически все программное обеспечение создается с применением разнообразных сторонних библиотек и фреймворков. Разработчик программного обеспечения в начале проектирования должен в первую очередь изучить существующие библиотеки для определения множества имеющихся компонентов, которые он сможет переиспользовать в своем проекте, не реализуя их заново. Основой для принятия решения при этом могут являться документация, предоставляемая авторами, и отзывы других разработчиков. К сожалению, на данный момент распространенной практикой является поставка повторно используемых компонентов “as is”, т. е. с минимальным объемом документации и поясняющих комментариев. Как правило, все ограничивается кратким описанием функций и форматов данных, составляющих интерфейс библиотеки, которое приводится на естественном языке. В некоторых случаях также описываются один или несколько упрощенных примеров ее использования. К наиболее существенным недостаткам подобного подхода относятся неполнота и неоднозначность спецификаций библиотек [1], что, несомненно, не может не сказываться на времени и качестве разработки. Для преодоления этих ограничений могут быть использованы спецификации библиотек, задаваемые с применением специализированных формальных языков. Помимо документирования, такие спецификации могут быть также использованы для проведения автоматизированной проверки соблюдения программистом правил применения библиотечных компонентов. Соответствующие подходы разрабатываются в лаборатории верификации и анализа программ кафедры компьютерных систем и программных технологий Санкт-Петербургского политехнического университета Петра Великого [2, 3].

Успешность применения подходов, базирующихся на спецификациях библиотек, напрямую зависит от наличия и качества таких спецификаций. Ввиду того, что авторы большинства библиотек вряд ли сами будут представлять формальные спецификации, необходимы другие механизмы, которые позволят, хотя бы частично, автоматизировать их создание.

Объектом исследования, представленного в настоящей статье, является опыт применения подходов, относящихся к направлениям Empirical

Software Engineering (эмпирическая программная инженерия) и Mining Software Repositories (извлечение данных из репозиториев), для решения задачи восстановления спецификаций программных библиотек путем обобщения пользовательского опыта, накопленного в результате реализации десятков тысяч программных проектов. Использование рассматриваемых подходов предполагает осуществление анализа программных проектов с открытым исходным кодом, в которых задействованы компоненты интересующей библиотеки, с применением методов статического и динамического анализа программного обеспечения. Результатом такого анализа являются полные или частичные модели библиотек, которые могут использоваться как для формирования на их основе формальных спецификаций, так и для документирования указанных библиотек.

Предметом обзора, представленного в статье, являются существующие подходы, которые потенциально могут быть адаптированы для автоматизированного извлечения спецификаций, заданных с использованием формализма, представленного в работе [2]. К отличительным особенностям этого формализма относится то, что он позволяет фиксировать правила применения библиотеки в максимально полном и детализированном виде, а также захватывать правила, характеризующие взаимосвязи, существующие между различными объектами в рамках библиотеки и состоянием самой библиотеки.

Все рассматриваемые в данном обзоре методы опираются на следующие предположения:

— исходный текст искомых библиотек не доступен для анализа;

— известны API (Application Programming Interface) библиотек и используемые типы данных;

— доступно для изучения существенное число программных проектов, лицензия которых не ограничивает доступ к исходному коду и позволяет запускать проект для исследовательских целей.

Виды формализмов для представления спецификаций

Существует несколько возможных подходов к определению спецификации программной библиотеки, отличающихся по виду данных, фиксируемых с помощью правил, включаемых в ее состав. При этом в большинстве рассматриваемых подходов, за исключением [2], сама библиотека не выделяется в отдельную сущность и ее поведение не описывается явно. Имеющиеся на данный момент модели правил можно классифицировать следующим образом:

1) совместно используемые множества элементов API [4–7];

2) ограничения на значения состояний целевого объекта и параметров вызываемых методов [8–10];

3) временные свойства [11–26];

4) комплексные модели [2, 27–29] (модели подходов CONTRACTOR++, SEKT, TEMI).

С помощью моделей *первой* группы фиксируется необходимость совместного использования элементов API, при этом последовательность обращений не учитывается. Элементами моделей данного вида могут выступать как переменные фиксированного типа [5], так и вызовы методов [4–7]. Необходимо отметить, что на данный момент рассматриваемая группа является единственной, которая поддерживает представление связей реализации и наследования [4, 7].

Правила, сформированные с использованием моделей *второй* группы, позволяют фиксировать аксиоматические предусловия и постусловия методов и инварианты классов. В качестве элементов в них используются переменные, вызовы методов и диапазоны значений для применяемых переменных и возвращаемых результатов [8, 10], осуществляемые проверки значений результатов вызовов взаимно упорядоченных методов и соответствующие вызовы [9]. Искомые отношения между элементами задаются с использованием расширяемых наборов шаблонов и логических операций первого порядка (в частности, формируемые правила могут быть условными [8, 9]).

Формализмы, лежащие в основе правил *третьей* группы, наиболее многочисленны и разнообразны. Они позволяют представлять последовательности вызовов методов, допустимых для совместного использования [13, 14, 16, 18, 22], а также тех, которые необходимо использовать совместно [11, 12, 15, 17, 19–21, 23–26]. При этом в рамках различных моделей методы могут являться как связанными по данным [13, 14, 16, 18, 22–24] или владеющему объекту [14–16], так и независимыми [11, 12, 17, 19–21, 25, 26].

Формализмы, применяемые для фиксации временных свойств, могут быть разделены по виду используемых базовых моделей следующим образом:

1) конечные автоматы (КА) [11–13, 15–20, 23, 25, 26, 30];

2) формулы темпоральной логики [14, 21];

3) отношения в виде частичных порядков [22];

4) модифицированные графы потока управления (Control Flow Graph — CFG) [24].

Для представления предполагаемого алгоритма использования элементов API с помощью моделей первой подгруппы используются детерминированные [11–13, 15–20, 23, 25, 26, 30] КА.

В качестве состояний такого автомата могут быть использованы участки исходного кода перед вызовом библиотечных методов внутри метода пользовательской программы [13, 16, 25], сами вызовы методов [19], а также состояния моделируемых абстрактных объектов [18, 20, 23, 26, 30]. Альтернативным решением является хранение «упрощенного» состояния целевого объекта или контекста метода, которые будут зависеть только от вызовов выбранных методов [11, 12, 15, 17].

Специальный вид состояний может быть введен также для фиксации ситуаций, приводящих к возникновению исключений [16]. Для описания последовательности вызовов во всех рассмотренных моделях используются переходы. В значительном числе подходов извлекаемые правила моделируются с использованием шаблонных КА, распознающих алфавит из двух [11, 12, 15, 23] или трех [20, 26] символов. Восстановление более сложных правил возможно путем конструирования составного КА, узлами которого будут являться автоматы элементарных правил, объединяемые последовательно [15, 26] или параллельно [26].

Формализмы *второй* подгруппы позволяют определять порядок выполнения вызовов с помощью формул логики будущего [14] и прошедшего [21] времени, в которых в качестве предикатов выступают соответствующие события. Последний вид формул, помимо определения порядка, позволяет также явным образом задавать асимметричные требования к совместному выполнению вызовов.

Представление правил с применением моделей *третьей* подгруппы [22] позволяет фиксировать временные свойства как иерархическую структуру, узлами которой являются вызовы методов, а связи отображают допустимый порядок между ними.

Правила, сформированные с применением шаблонов *четвертой* подгруппы [24], позволяют фиксировать временные свойства в наиболее развернутом виде (помимо последовательности связанных по данным вызовов, фиксируются управляющие конструкции, используемые совместно с ними, а также то, через какие параметры они связаны).

Наиболее часто встречающимся решением является моделирование временных свойств с использованием формализмов первой подгруппы. Необходимо отметить, что модели, поддерживающие представление временных свойств в виде КА свободной формы, являются более мощными, чем модели с использованием шаблонов и методов композиции. Была показана принципиальная невозможность фиксации некоторых видов правил с применением последних [20], однако в настоящее время ввиду сравнительной простоты реализации применение таких моделей остается одним из наиболее часто используемых решений.

Существенно менее многочисленны составляющие *четвертую* группу комплексные модели, позволяющие одновременно фиксировать различные виды свойств [2, 27–29], таких как временные свойства и ограничения по данным. Большая часть моделей, принадлежащих к данной группе, основывается на использовании расширенных КА, в которых с помощью меток над переходами могут описываться ограничения, описывающие предполагаемые значения параметров или состояния целевого объекта для подходов [29] (SEKT), [28] (для подхода, предложенного в работе [2], также предлагается описывать влияние на окружение), а также факт подтвержденности перехода для [29] (TEMI). Используемые автоматы могут быть детерминированными [2], [29] (CONTRACTOR++) или недетерминированными [28, 29] (SEKT, TEMI); при этом все рассматриваемые формализмы позволяют захватывать разрешенные последовательности вызовов и соответствующие им ограничения на данные в свободной форме. В работе [27] был предложен специальный вид моделей, основанный на сопоставлении абстрактным переменным, используемым совместно, множеств вызовов, в которых они используются в том или ином качестве, и осуществляемых над ними проверок. Модели такого вида позволяют выявлять стандартные комплексные правила в свободной форме.

Наиболее подробное описание поведения библиотеки среди рассматриваемых формализмов в виде совокупности КА свободной формы позволяет зафиксировать модель, представленную в работе [2]. В рамках рассматриваемого подхода для задания спецификации предлагается использовать иерархию взаимосвязанных расширенных КА. Автомат верхнего уровня используется для представления состояния библиотеки в целом, в то время как автоматы нижележащих уровней, являющиеся дочерними по отношению к нему, применяются для описания стандартных жизненных циклов отдельных объектов. Взаимодействие нескольких объектов в данной модели может быть представлено неявно с использованием КА верхнего уровня. Помимо временных свойств и ограничений на данные, описание методов библиотеки, осуществляемое с применением формализма, должно также содержать информацию о выполняемых ими наборах элементарных семантических операций [3].

Цель исследования, представленного в данной статье, составляет поиск и анализ перспективных статических методов автоматизированного извлечения спецификаций программных библиотек на основе рассмотренного формализма, позволяющих получать наиболее полные и точные результаты.

Статические методы извлечения спецификаций

Обобщенный алгоритм выявления правил с помощью методов анализа исходного кода [4–7, 9, 11, 13, 14, 16, 17, 22–25, 27] можно представить в виде последовательности шагов:

1) выбор контекста поиска паттернов;

2) формирование обобщенных экземпляров контекста поиска;

3) кластеризация экземпляров контекста поиска;

4) извлечение «транзакций»;

5) предварительная обработка «транзакций»;

6) поиск правил;

7) предоставление результатов пользователю.

В качестве контекста могут использоваться классы [4, 7], методы (преимущественно выбираются именно они [5, 9, 17, 22–25, 27]) и переменные фиксированного библиотечного типа [6, 11, 13, 14, 16]. При выборе контекста уровня классов выделяются свойства, связанные как с использованием механизмов наследования, так и с вызовом методов. При этом в качестве единицы, использующей метод, рассматривается весь класс. Если для осуществления поиска выбран контекст, состоящий из методов «клиентской» программы или переменных фиксированного типа, то предлагаются стратегии формирования «статических трасс». Такие «трассы» генерируются путем реализации различных стратегий обхода графов потока управления (в том числе могут быть учтены цепочки вызовов методов, приводящие к генерации исключений [23]); они содержат информацию о вызываемых библиотечных методах, зависимостях по данным между ними, а также об управляющих структурах.

Сформированные описания могут быть далее уточнены (разбиты на кластеры, составляющие изолированные множества описаний, в которых будет осуществляться поиск) следующим образом:

— совместно вызываемые методы и осуществляемые проверки [27], связанные по данным (формирующие «сценарий» или «модель использования объекта») [6, 11, 13, 14, 16, 22–24, 27];

— методы, используемые при реализации одного и того же абстрактного метода используемого фреймворка [6, 25];

— совокупность обращений к методам API, названия которых отличаются только суффиксами (например, использование пары суффиксов lock/unlock будет говорить о принадлежности к одной и той же латентной спецификации) [17];

— множество осуществляемых до и после вызова конкретного метода API проверок [9].

В сформированной совокупности «транзакций», состоящих из обращений к элементам внешнего интерфейса, с применением методов статистического анализа осуществляется поиск частых множеств.

Полученный набор фильтруется для снижения ресурсоемкости выполнения следующего шага с применением эвристик, разработанных на основе знания о языке или компоненте. Например, из них могут быть удалены элементы, являющиеся вызовами стандартных функций, таких как *equals()* или *add()*, или цепочки get-методов, позволяющие обратиться к конкретному полю сложного объекта [11]. Также предложены метрики «доверия» для различных фрагментов кода, позволяющие рассчитать их «вес» (относительную важность) на основе информации о процессе разработки, извлекаемой из репозитория [23].

После фильтрации осуществляется непосредственно поиск правил использования элементов анализируемой библиотеки. Все используемые методы статического анализа можно разделить по виду применяемых алгоритмов статистического анализа на следующие группы:

1) методы поиска частых множеств [4–6, 7, 9, 11, 13, 14, 16, 23, 27];

2) методы поиска частых частичных порядков [22];

3) методы поиска частых последовательностей [17, 25, 27];

4) методы поиска частых изоморфных подграфов [24].

Порог валидности генерируемых правил задается пользователем вручную через указание минимальных значений поддержки (support; отношение количества «транзакций», содержащих посылку и следствие, к общему числу «транзакций») и достоверности (confidence; отношение количества «транзакций», содержащих посылку и следствие, к количеству «транзакций», содержащих только посылку) или аналогичных параметров.

Основные алгоритмы, используемые для поиска *частых множеств* неупорядоченных элементов, это: «наивный» поиск часто встречающихся наборов обращений к элементам API [6, 9]; классический алгоритм Apriori и его модифицированная версия, поддерживающая работу с наследованием [4, 11]; поиск максимальных множеств [5, 7, 27], а также анализ формальных понятий [14]. На их основе были разработаны подходы, позволяющие извлекать КА, распознающие алфавит из фиксированного количества символов, т. е. жестко ограниченное подмножество частых последовательностей. Для достижения данной цели могут быть использованы методы «наивного» подсчета частоты совместного использования пар последовательно идущих вызовов методов [23] или осуществлен переход к «укрупнению»

рассматриваемых элементов транзакций в сочетании с использованием стандартных методов поиска частых множеств неупорядоченных элементов [13, 16]. В последнем случае каждый элемент будет представлять собой упорядоченный набор обращений.

Для поиска *частых частичных порядков* (frequent partial order), описывающих стандартные последовательности вызовов методов API, был предложен алгоритм FRECPO [22]. Его можно рассматривать как модифицированную версию алгоритма Apriori, с помощью которой можно извлекать упорядоченные множества элементов без использования шаблонов.

Методы поиска *частых последовательностей* являются расширением методов группы поиска частых множеств. С их помощью может быть осуществлено извлечение часто встречающихся упорядоченных последовательностей вызовов произвольной длины.

Поиск спецификаций программных библиотек в виде *частых направленных ациклических подграфов* предполагает выявление изоморфных подграфов потоков управления в анализируемых методах клиентских программ [24]. Ключевыми достоинствами такого подхода являются возможность фиксации информации об управляющих конструкциях, часто применяемых совместно с компонентами исследуемой библиотеки, и возможность осуществления непосредственно поиска сложных правил в свободной форме.

Для удобства применения найденные правила или зафиксированные с их применением ошибки при отображении фильтруются и сортируются в соответствии с конкретной решаемой задачей. Так, например, при построении рекомендательной системы используются вероятностные оценки значимости правила [4, 7, 23], а также может быть учтен контекст осуществляемого вызова [25], при выявлении ошибок — вероятность того, что найденный фрагмент кода является ошибочным [5, 11].

В связи с тем, что решаемые задачи являются принципиально разными, сравнить ресурсоемкость подходов и качество получаемых с их применением результатов довольно затруднительно, что отмечается, например, и в обзоре [31]. Часть статей описывает только использование инструментов на уровне пользователя или содержит анализ отдельных результатов [5, 7, 14, 16, 17, 22], часть содержит более подробную информацию об общем количестве найденных истинных и ложных, а также о количестве пропущенных правил [4, 6, 9, 11, 13, 23–25, 27]; некоторые статьи содержат также информацию о времени поиска и используемых аппаратных средствах [5, 13, 14, 16, 27]. Обобщая, несмотря на недостаток систематизированной информации, можно выделить следующие общие свойства существующих решений.

Большое количество извлекаемых правил. В обзорах, которые содержат информацию об общем количестве найденных паттернов, отображены результаты от десятков [9, 11, 23, 25] до тысяч, десятков тысяч [4, 5, 16, 27, 24] и миллионов [13] правил. Для использования извлеченных шаблонов конечными пользователями требуется дополнительная автоматизированная постобработка.

Разные показатели точности получаемых результатов. Для значительного количества рассмотренных работ, связанных с автоматизированным поиском ошибок на основе извлекаемых правил, оценка рассматриваемых критериев может быть произведена опосредованно на основе информации о количестве ошибочно выявленных правил [9, 27] (так же, как и для систем выявления правил использования компонентов библиотеки [11, 22]) или соотношении истинных и ложных нарушений [5, 9, 13, 14, 16, 17, 24], распознанных с их использованием. В случае же если основной целью разрабатываемого подхода является построение рекомендательной системы [4, 6, 7, 25], для такой оценки могут быть использованы данные о количестве корректно и ошибочно предсказанных вызовов (при этом под ошибочно предсказанными вызовами понимаются вызовы, не соответствующие реально существующим правилам использования компонентов библиотеки [6], а не вызовы, относящиеся к правилам, не востребованным пользователями [25]).

Информация о показателях точности результатов, получаемых с использованием методов статического анализа, представлена в табл. 1 и 2.

Для подходов [4, 7], позволяющих извлекать правила в виде множеств совместно используемых элементов API, информация о качестве результатов не приводится. Для остальных подходов для извлечения правил данного вида точность получаемых результатов варьируется от сравнительно низкого значения 16 % [5] до 98 % [6]. Однако необходимо отметить, что высокая точность искомых правил, получаемых с помощью метода, представленного в работе [6], обусловливается жестким ограничением на качество исходного кода (ошибка в применении правила согласно введенным ограничениям может заключаться не более чем в наличии или отсутствии вызова только одного метода). Очевидно, что последнее требование является невыполнимым в случае работы с исходным кодом реальных, а не лабораторных проектов.

Единственный среди рассмотренных подходов, основанных на применении методов статического анализа, подход [9], позволяющий получать ограничения на значения состояний целевого объекта и параметры вызываемых методов, напротив,

■ ***Таблица 1.*** Подходы для извлечения совместно используемых элементов API и ограничений на значения состояний целевого объекта и параметры вызываемых методов

■ ***Table 1.*** Approaches to mining of rules containing jointly used library elements and rules containing predicates for allowed target object's states and method parameters

Работа	Элементы моделей	Вид статистических методов	Точность, %
Совместно используемые элементы API			
[4]	Элементы базовых двухэлементных множеств — вызовы методов, наследование от классов и реализация интерфейсов	Поиск частых множеств	–
[5]	Аналогично [4]	То же	16
[6]	Элементы множеств — совместно используемые вызовы	– ″ –	98
[7]	Аналогично [4]	– ″ –	–
Ограничения на значения состояний целевого объекта и параметры вызываемых методов			
[9]	Множества проверок, осуществляемых на параметрах и результатах вызова методов, с использованием шаблонов	Поиск частых множеств	67

■ ***Таблица 2.*** Подходы для извлечения временных свойств

■ ***Table 2.*** Approaches to mining of rules containing temporal properties

Работа	Вид временных свойств	Элементы моделей	Точность, %
[11]	КА	Пары вызовов методов, используемых совместно на одном объекте	66
[13]			22
[16]			48
[14]	Формулы CTL	Предикаты, описывающие вызовы методов, которые требуется совершить перед вызовом данного	54
[17]	КА	MUST- и MAY-правила осуществления вызовов библиотечных методов, выведенные на основе пользовательских шаблонов, задаваемых для конкретных типов данных	64
[22]	Отношения частичного порядка	Правила, описывающие последовательности вызовов методов без использования шаблонов, могут быть адаптированы для включения вызовов на объектах разных классов	–
[23]	КА	Пары вызовов методов; могут быть адаптированы для включения вызовов на объектах разных классов	95 («точный»), 31 («нормальный»)
[24]	Модифицированный граф потока управления	Последовательности вызовов методов на различных объектах и управляющие конструкции	66

заслуживает внимания. С его помощью можно извлечь информацию о множестве проверок, которые необходимо осуществить до и после вызова библиотечного метода, причем извлекаемые правила формируются с помощью формул булевой логики. Средняя точность получаемых результатов составляет 67 %. Данный подход может быть адаптирован для восстановления как стандартных ограничений на данные, так и временных свойств.

Однако в рамках сформулированной цели исследования наибольший интерес потенциально представляют показатели качества для методов статического анализа, позволяющих извлекать правила в виде временных свойств (см. табл. 2), а также комплексные правила.

Для подходов [11, 13, 14, 16, 17, 23, 24], предназначенных для извлечения временных свойств, точность получаемых результатов будет варьироваться незначительно, и в среднем будет составлять около 60 %. Необходимо отметить, что для заметного количества подходов [13, 14, 16, 24] при оценке качества учитывались как зафиксированные с использованием полученных правил реальные ошибки, так и «code smells» (потенциально некорректные или неверно оформленные фрагменты кода), для которых не выполняются статистически выявленные правила, но которые при этом не являются ошибочными. Показатели качества для подходов с применением шаблонов и без их использования являются близкими. Наилучших результатов среди всех обозреваемых подходов (95 % извлеченных правил будут являться истинными) позволяет достичь использование подхода, учитывающего при поиске правил значение метрик степени важности того или иного фрагмента используемого кода (в том числе рассчитываемых с использованием артефактов разработки), который был представлен в работе [23], а именно поддерживаемой им стратегии «точного» поиска, при котором осуществляется максимизация данного критерия, в противовес «нормальному поиску», при котором одновременно учитываются и полнота, и точность результата. Платой за выигрыш в точности здесь будет снижение полноты более чем в три раза. Необходимо отметить, что полнота в данном случае оценивалась относительно правил, извлеченных с использованием инструмента WnMiner, что могло повлиять на конечный результат. Кроме снижения полноты, к недостаткам данного подхода относится ограниченность множества правил, которые можно зафиксировать с применением лежащей в его основе модели: несмотря на предоставляемую возможность захвата правил для объектов разных классов, с ее использованием может быть зафиксирована только последовательность длиной в два вызова, что без дополнительной обработки существенно ограничивает множество распознаваемых правил. Для исследований [22, 25] данные, которые могут быть использованы для статистически значимой количественной оценки результатов, не представлены. Среди подходов без использования поисковых шаблонов наилучших показателей качества извлекаемых спецификаций позволяет достичь подход, предложенный в работе [24], который базируется на менее распространенной модели представления правил в виде модифицированного графа потока управления. Для всех подходов, за исключением представленного в работе [25], в которых спецификации фиксируются в виде КА, неотъемлемой чертой является использование шаблонов (подход, рассмотренный в работе [25], позволяет захватывать множество правил, представимых в виде цепочек последовательных вызовов произвольной длины).

Информация о качестве результатов для единственного из обозреваемых подхода [27], который позволяет фиксировать комплексные правила (и при этом делать это без использования шаблонов), приводится на основе анализа одного проекта.

Заключение

Целью проведенного исследования являлся поиск перспективных методов автоматизированного извлечения спецификаций для формализма описания библиотек, предложенного в работе [2]. В связи с видом выбранной (комплексной) модели наибольший интерес представляют методы, позволяющие восстанавливать временные свойства и комплексные правила, описывающие процесс взаимодействия с библиотечными компонентами.

Рассматриваемые в обзоре методы, основанные на алгоритмах статического анализа, позволяют захватывать только стандартные (частые) последовательности вызовов. Установлено, что точность правил, фиксирующих ограничения на данные в «свободной форме» (без использования шаблонов), будет составлять порядка 70 %. Правила, описывающие временные свойства, которые могут быть извлечены с использованием методов статического анализа, будут обладать средней точностью 60 %. При этом правила, основанные на использовании конечно-автоматного формализма, с помощью методов рассматриваемой группы могут быть восстановлены только с использованием шаблонов. Данные о систематической оценке качества для комплексных правил, извлекаемых с использованием рассматриваемых методов, в проанализированных работах не приводятся.

Продолжение обзора, в котором приводятся результаты исследования подходов к извлечению спецификаций программных библиотек методами динамического анализа, а также осуществляется сравнение подходов различных видов для решения поставленной задачи извлечения комплексных правил, является предметом отдельной статьи.

Работа частично поддержана стипендиальной программой компании «Сименс» в СПбПУ.

Литература

1. **Uddin G., Robillard M. P.** How API Documentation Fails // IEEE Software. 2015. Vol. 32. P. 68–75.
2. **Ицыксон В. М.** Формализм и языковые инструменты для описания семантики программных библиотек // Моделирование и анализ информационных систем. 2016. № 23. С. 754–766.
3. **Ицыксон В. М., Зозуля А. В.** Формализм для описания частичных спецификаций компонентов программного окружения // Научно-технические ведомости СПбГПУ. Секция «Информатика. Телекоммуникации. Управление». 2011. № 4(128). С. 81–90.
4. **Michail A.** Data Mining Library Reuse Patterns using Generalized Association Rules // Proc. of the 22nd Intern. Conf. on Software Engineering. 2000. P. 167–176.
5. **Zhou Y., Li Z.** PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code // Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. 2005. P. 306–315.
6. **Monperrus M., Bruch M., and Mezini M.** Detecting Missing Method Calls in Object-Oriented Software // Proc. of the 24th European Conf. on Object-Oriented Programming, ECOOP 2010, Maribor, Slovenia, June 21–25, 2010. Lecture Notes in Computer Science, 2010. Vol. 6183. P. 2–25.
7. **Bruch M., Schaffer T., Mezini M.** FrUiT: IDE Support for Framework Understanding // Proc. of the 2006 OOPSLA Workshop on Eclipse Technology eXchange. 2006. P. 55–59.
8. **Henkel J., Diwan A.** Discovering Algebraic Specifications from Java Classes // Proc. of the 17th European Conf. on Object-Oriented Programming, ECOOP 2003, Darmstadt, Germany, July 21–25, 2003. Lecture Notes in Computer Science, 2003. Vol. 2743. P. 431–456.
9. **Thummalapenta S., Xie T.** Alattin: Mining Alternative Patterns for Defect Detection // Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering. 2009. P. 293–323.
10. **Ernst M. D., Perkins J. H., Guo P. J., McCamant S., Pacheco C., Tschantz M. S., Xiao C.** The Daikon System for Dynamic Detection of Likely Invariants // Science of Computer Programming. 2007. Vol. 69. Iss. 1–3. P. 35–45.
11. **Livshits B., Zimmerman T.** DynaMine: finding Common Error Patterns by Mining Software Revision Histories // Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. 2005. P. 296–305.
12. **Gabel M., Su Z.** Online Inference and Enforcement of Temporal Properties // Proc. of the 32nd ACM/IEEE Intern. Conf. on Software Engineering. 2010. Vol. 1. P. 15–24.
13. **Gruska N., Wasylkowski A., Zeller A.** Learning from 6,000 Projects: Lightweight Cross-Project Anomaly Detection // Proc. of the 19th Intern. Symp. on Software Testing and Analysis. 2010. P. 119–130.
14. **Wasylkowski A., Zeller A.** Mining Temporal Specifications from Object usage // Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering. 2009. P. 295–306.
15. **Yang J., Evans D., Bhardwaj D., Bhat T., Das M.** Perracotta: Mining Temporal API Rules from Imperfect Traces // Proc. of the 28th Intern. Conf. on Software Engineering. 2006. P. 282–291.
16. **Wasylkowski A., Zeller A., Lindig C.** Detecting Object usage Anomalies // Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2007. P. 35–44.
17. **Engler D., Chen D. Y., Hallem S., Chou A., Chelf B.** Bugs as Deviant Behavior // Proc. of the 18th ACM Symp. on Operating Systems Principles. 2001. P. 57–72.
18. **Ammons G., Bodik R., Larus J. R.** Mining Specifications// Proc. of the 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2002. P. 4–16.
19. **Pradel M., Gross T. L.** Automatic Generation of Object usage Specifications from Large Method Traces // Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering. 2009. P. 371–382.
20. **Gabel M., Su Z.** Symbolic Mining of Temporal Specifications // Proc. of the 30th Intern. Conf. on Software Engineering. 2008. P. 51–60.
21. **Lo D., Khoo S., Liu C.** Mining Past-Time Temporal Rules from Execution Traces // Proc. of the 2008 Intern. Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT Intern. Symp. on Software Testing and Analysis (ISSTA 2008). 2008. P. 50–56.
22. **Acharya M., Xie T., Pei J., Xu J.** Mining API Patterns as Partial Orders from Source Code: From usage Scenarios to Specifications // Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2007. P. 25–34.
23. **Goues C., Weimer W.** Specification Mining with Few False Positives // Proc. of the 15th Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems: Held as Part of the Joint European Conf. on Theory and Practice of Software. 2009. P. 292–306.
24. **Nguyen T. T., Nguyen H. A., Pham N. H., Al-Kofahi J. M., Nguyen T. N.** Graph-based Mining of Multiple Object usage Patterns // Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2009. P. 383–392.
25. **Zhong H., Xie T., Zhang L., Pei J., Mei H.** MAPO: Mining and Recommending API usage Patterns // Proc. of the 23rd European Conf. on Object-Oriented Programming, ECOOP 2009, Genoa, Italy, July 6–10, 2009.

Lecture Notes in Computer Science, 2009. Vol. 5653. P. 318–343.
26. **Gabel M., Su Z.** Javert: Fully Automatic Mining of General Temporal Properties from Dynamic Traces // Proc. of the 16th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. 2008. P. 339–349.
27. **Ramanathan M., Grama A., Jaganathan S.** Static Inference with Predicate Mining // Proc. of the 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2007. P. 123–134.
28. **Lorenzoli D., Mariani L., Pezze M.** Automatic Generation of Software Behavioral Models // Proc. of the 30th Intern. Conf. on Software Engineering. 2008. P. 501–510.
29. **Krka I., Medvidovic N., Brun Y.** Automatic Mining of Specifications from Invocation Traces and Method Invariants // Proc. of the 22nd ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. 2014. P. 178–189.
30. **Alur R., Cerny P., Madhusudan P., Nam W.** Synthesis of Interface Specifications for Java Classes // Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2005. P. 98–109.
31. **Robillard M. P., Bodden E., Kawrykow D., Mezini M., Ratchford T.** Automated API Property Inference Techniques // IEEE Transactions on Software Engineering. 2013. Vol. 39. Iss. 5. P. 613–637.

UDC 004.4′2
doi:10.15217/issn1684-8853.2017.6.66

Survey of Static Methods for Partial Software Library Specifications Extraction

Egorova I. S.[a], Post-Graduate Student, is.egorova@mail.ru
Itsykson V. M.[a], PhD, Tech., Associate Professor, vlad@icc.spbstu.ru
[a]Peter the Great St. Petersburg Polytechnic University, 29, Polytechnicheskaya St., 195251, Saint-Petersburg, Russian Federation

Introduction: Wide usage of third-party software libraries and frameworks for software development along with the lack of their precise documentation actualize the problem of creating formal software library specifications. Recovery of such specifications could be performed with the application of successfully developed open-source projects. **Purpose:** Analysis and classification of prospective approaches to automated extraction of the most expressive complex software library specifications based on the methods of static code analysis. **Results:** Various approaches used to describe library components have been reviewed and compared. We have discussed the ways of deriving standard (frequent) specifications using static analysis methods, and recovered the common algorithm they share. It is found out that most methods based on static analysis of the code presume the usage of templates to define a set of rules. Specifications frequently recovered in this way should only describe a lifecycle for objects of a single class. The quality of the resulting specifications is usually quite low so far. To obtain better results, complex specifications should be used whose recovery methods are currently few in numbers and not systematized.

Keywords — Software Library Specification, Formal Specification, Specification Extraction, Extended Finite State Machine, Temporal Properties, Behavioral Software Library Model, Complex Rules, Static Analysis.

References

1. Uddin G., Robillard M. P. How API Documentation Fails. *IEEE Software*, 2015, vol. 32, pp. 68–75.
2. Itsykson V. M. The Formalism and Language Tools for Semantics Specification of Software Libraries. *Modelirovanie i analiz informatsionnykh sistem* [Modeling and Analysis of Information Systems], 2016, no. 23, pp. 754–766 (In Russian).
3. Itsykson V. M., Zozulya A. V. Formalism for Description of Software Components Partial Descriptions. *Nauchno-tekhnicheskie vedomosti SPbGPU. Sektsiia "Informatika. Telekommunikatsii. Upravlenie"* [Scientific Journal of SPBSPU. Section "Computer Science. Telecommunication. Control"], 2011, no. 4(128), pp. 81–90 (In Russian).
4. Michail A. Data Mining Library Reuse Patterns using Generalized Association Rules. *Proc. of the 22nd Intern. Conf. on Software Engineering*, 2000, pp. 167–176.
5. Zhou Y., Li Z. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. *Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering*, 2005, pp. 306–315.
6. Monperrus M., Bruch M., and Mezini M. Detecting Missing Method Calls in Object-Oriented Software. *Proc. of the 24th European Conf. on Object-Oriented Programming, ECOOP 2010, Maribor, Slovenia, June 21–25, 2010*. Lecture Notes in Computer Science, 2010, vol. 6183, pp. 2–25.
7. Bruch M., Schaffer T., Mezini M. FrUiT: IDE Support for Framework Understanding. *Proc. of the 2006 OOPSLA Workshop on Eclipse Technology eXchange*, 2006, pp. 55–59.
8. Henkel J., Diwan A. Discovering Algebraic Specifications from Java Classes. *Proc. of the 17th European Conf. on Object-Oriented Programming, ECOOP 2003, Darmstadt, Germany, July 21–25, 2003*. Lecture Notes in Computer Science, 2003, vol. 2743, pp. 431–456.
9. Thummalapenta S., Xie T. Alattin: Mining Alternative Patterns for Defect Detection. *Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering*, 2009, pp. 293–323.
10. Ernst M. D., Perkins J. H., Guo P. J., McCamant S., Pacheco C., Tschantz M. S., Xiao C. The Daikon System for Dynamic Detection of Likely Invariants. *Science of Computer Programming*, 2007, vol. 69, iss. 1–3, pp. 35–45.
11. Livshits B., Zimmerman T. DynaMine: finding Common Error Patterns by Mining Software Revision Histories. *Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering*, 2005, pp. 296–305.
12. Gabel M., Su Z. Online Inference and Enforcement of Temporal Properties. *Proc. of the 32nd ACM/IEEE Intern. Conf. on Software Engineering*, 2010, vol. 1, pp. 15–24.
13. Gruska N., Wasylkowski A., Zeller A. Learning from 6,000 Projects: Lightweight Cross-Project Anomaly Detection. *Proc. of the 19th Intern. Symp. on Software Testing and Analysis*, 2010, pp. 119–130.
14. Wasylkowski A., Zeller A. Mining Temporal Specifications from Object usage. *Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering*, 2009, pp. 295–306.
15. Yang J., Evans D., Bhardwaj D., Bhat T., Das M. Perracotta: Mining Temporal API Rules from Imperfect Traces. *Proc.*

of the 28th Intern. Conf. on Software Engineering, 2006, pp. 282–291.
16. Wasylkowski A., Zeller A., Lindig C. Detecting Object usage Anomalies. *Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering*, 2007, pp. 35–44.
17. Engler D., Chen D. Y., Hallem S., Chou A., Chelf B. Bugs as Deviant Behavior. *Proc. of the 18th ACM Symp. on Operating Systems Principles*, 2001, pp. 57–72.
18. Ammons G., Bodik R., Larus J. R. Mining Specifications. *Proc. of the 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, 2002, pp. 4–16.
19. Pradel M., Gross T. L. Automatic Generation of Object usage Specifications from Large Method Traces. *Proc. of the 2009 IEEE/ACM Intern. Conf. on Automated Software Engineering*, 2009, pp. 371–382.
20. Gabel M., Su Z. Symbolic Mining of Temporal Specifications. *Proc. of the 30th Intern. Conf. on Software Engineering*, 2008, pp. 51–60.
21. Lo D., Khoo S., Liu C. Mining Past-Time Temporal Rules from Execution Traces. *Proc. of the 2008 Intern. Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT Intern. Symp. on Software Testing and Analysis (ISSTA 2008)*, 2008, pp. 50–56.
22. Acharya M., Xie T., Pei J., Xu J. Mining API Patterns as Partial Orders from Source Code: From usage Scenarios to Specifications. *Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering*, 2007, pp. 25–34.
23. Goues C., Weimer W. Specification Mining with Few False Positives. *Proc. of the 15th Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems: Held as Part of the Joint European Confe. on Theory and Practice of Software*, 2009, pp. 292–306.
24. Nguyen T. T., Nguyen H. A., Pham N. H., Al-Kofahi J. M., Nguyen T. N. Graph-based Mining of Multiple Object usage Patterns. *Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering*, 2009, pp. 383–392.
25. Zhong H., Xie T., Zhang L., Pei J., Mei H. MAPO: Mining and Recommending API usage Patterns. *Proc. of the 23rd European Conf. on Object-Oriented Programming, ECOOP 2009, Genoa, Italy, July 6–10, 2009*. Lecture Notes in Computer Science, 2009, vol. 5653, pp. 318–343.
26. Gabel M., Su Z. Javert: Fully Automatic Mining of General Temporal Properties from Dynamic Traces. *Proc. of the 16th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering*, 2008, pp. 339–349.
27. Ramanathan M., Grama A., Jaganathan S. Static Inference with Predicate Mining. *Proc. of the 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2007, pp. 123–134.
28. Lorenzoli D., Mariani L., Pezze M. Automatic Generation of Software Behavioral Models. *Proc. of the 30th Intern. Conf. on Software Engineering*, 2008, pp. 501–510.
29. Krka I., Medvidovic N., Brun Y. Automatic Mining of Specifications from Invocation Traces and Method Invariants. *Proc. of the 22nd ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering*, 2014, pp. 178–189.
30. Alur R., Cerny P., Madhusudan P., Nam W. Synthesis of Interface Specifications for Java Classes. *Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, 2005, pp. 98–109.
31. Robillard M. P., Bodden E., Kawrykow D., Mezini M., Ratchford T. Automated API Property Inference Techniques. *IEEE Transactions on Software Engineering*, 2013, vol. 39, iss. 5, pp. 613–637.