УДК 004.441

СТАТИЧЕСКИЙ АНАЛИЗ СЕМАНТИЧЕСКИХ СВЯЗЕЙ МУЛЬТИЯЗЫКОВЫХ ТЕКСТОВ ПРОГРАММ

Орловский М.Ю.[1] (магистрант)
Научный руководитель – доцент, кандидат технических наук, Логинов И.П.[1]
1 - Университет ИТМО
email: uberdever@niuitmo.ru

Аннотация
В современной индустрии разработки программного обеспечения нередким является применение нескольких языков программирования в одном программном проекте. Такой подход порождает проблему учета согласованности компонентов, реализованных на разных языках и применяемых в составе одной программной системы. Предложенный метод статического анализа предполагает семантический анализ связей программных компонентов для дальнейшего использования такого анализа в инструментальных средствах. Особенностями анализа являются гибкость в объеме анализа и универсальность относительно используемых языков.

Ключевые слова
Статический анализ, мультиязыковой анализ, инструментальные средства, семантический анализ, семантические сети, среды разработки.

Современные программные проекты, в отличие от многих программных проектов прошлого, гораздо чаще состоят из набора разных (порой разительно) технологических решений, предназначенных для решения определенного круга задач. В рамках данной статьи интересно то множество технологических решений, использующее собственные языки предметной области (DSL). В качестве примеров можно привести следующие языки, нередко фигурирующие в составе современных программных проектов:

- язык разметки HTML в составе проекта, использующего ASP фреймворк;
- язык скриптов командной строки в составе проекта, использующего язык C;
- язык запросов SQL в составе проекта, использующего Python и фреймворк Flask;
- язык препроцессора в составе файла исходного кода, реализованного на C++.

Заключительный пункт списка примеров приведен для того, чтобы показать характер связи различных технологий — разным языкам необязательно даже находится в раздельных файлах или модулях, нередки случаи полноценного переплетения различных синтаксисов и семантик. Так как используемые в проекте технологии, за редким исключением, практически всегда семантически связаны между собой, то возникает необходимость поддержания согласованности этих компонентов системы. В рамках статьи рассматривается проблема поддержания согласованности технологий, использующих разные языки программирования.

Особенности взаимодействия различных языков обычно затрудняют процесс разработки и при кодировании или рефакторинге нередки случаи нарушения согласованности. Как правило, такую несогласованность нельзя выявить до проведения процесса отладки или тестирования.

Однако, исходя из особенностей применяемых языков программирования часто возникает возможность применения методов статического анализа связности языковых конструкций между различными программными модулями, реализованными на разных языках программирования. Несмотря на это, текущие средства анализа являются недостаточными для программных проектов большого объема [1]. Также, стоит отметить проприетарную природу многих средств анализа и их неразрывную связь с другими инструментальными средствами, что затрудняет интеграцию таких анализаторов и адаптацию под конкретный проект. И несмотря на наличие открытых средств для статического мультиязыкового анализа, их применимость всё равно ограничена реализуемым стеком технологий и извлекаемой семантической информацией [2].

Основная проблема мультиязыкового анализа, конечно, заключается в большом разнообразии используемых технологий, но также стоит учесть различные сценарии использования такого рода анализа. Это также влияет на сложность анализаторов и их возможности в части улучшения процесса разработки.

В рамках данной статьи производится нестрогое разделение исходя из особенностей методов анализа.

По количеству обрабатываемой информации анализаторы можно разделить на:

1) анализаторы отдельных фрагментов ПК или файлов;
2) анализаторы групп файлов или групп фрагментов ПК;
3) анализаторы программных проектов, являющихся обособленными единицами программного комплекса.

Данное разделение не только позволяет оценить объемы программного кода, который подвергается анализу, но и правдивость и полноту анализа. Правдивость (англ. soundness) препятствует выдаче ложноотрицательных суждений. Полнота (англ. completeness) препятствует выдаче ложноположительных суждений [3]. Очевидно, что анализаторы из п. 1) редко являются полностью полными, а анализаторы из п. 2) и п. 3) полностью правдивыми.

По быстродействию анализаторы можно разделить на следующие категории:

1) быстрые или моментальные (задержка в 0–3 секунды);
2) медленные, но не прерывающие рабочий процесс (от 3 секунд до минуты);
3) очень медленные, прерывающие рабочий процесс (от минуты до суток).

Такое разделение позволяет оценить возможность интеграции анализаторов в рабочий процесс разработчика. Очевидно, что анализаторы из п. 1) предпочтительны в таких инструментах разработки как IDE и линтеры, когда как анализаторы из п. 2) и п. 3) не могут быть использованы в полной мере в процессе кодирования и рефакторинга и больше подходят для использования в иное время (например, в процессе сборки).

Исходя из вышеописанных сценариев использования, разработка универсального и практичного мультиязыкового анализатора представляется практически невыполнимой задачей. Поэтому, необходимо обозначить рамки полноты и правдивости анализа, а также рассмотреть лишь определенный круг сценариев использования. В рамках данной статьи рассматривается метод анализа, ориентированный на правдивость и быстроту, хотя также стоит заметить, что также имеется возможность обеспечить достаточную для практических нужд полноту анализа ценой времени и сложности самого анализа. Естественное предназначение анализа, описанного в статье — интеграция в инструментальные средства разработки (например, как плагин в среду разработки) для упрощения процесса разработки и рефакторинга.

Исходя из сложности поставленной задачи, было решено применять наиболее высокий уровень абстракции при представлении анализируемой информации. Таким образом, основная идея метода заключается в использовании унифицированного представления информации о программных компонентах для последующего осуществления семантического связывания этих компонентов. Такие представления являются по своей природе семантическими узлами, а процесс связывания — процессом создания семантической сети.

Таким образом, каждый анализируемый программный компонент имеет информационный узел, представляющий следующую структуру:

- позиция компонента в исходном коде (строка, столбец, файл);
- тип информации о компоненте;
- атрибуты, выявленные при анализе компонента;
- семантическая роль компонента.

В данной структуре ключевыми являются семантическая роль компонента и его атрибуты. Так как процесс связывания узлов полностью отделен от процесса получения этих узлов, роль компонента позволяет обеспечить:

- необходимую семантику анализа;
- достаточную полноту анализа;
- достаточную правдивость анализа.

Необходимая семантика может быть выстроена путем правильного моделирования информации об исходной технологии (и её языке) через атрибуты и семантические роли компонентов. Полнота анализа может быть обеспечена расширением множества

семантических ролей компонентов, что позволит получать больше информации о компоненте. Правдивость анализа может быть обеспечена заданием большего числа атрибутов, выявляемых при анализе компонента, что повысит точность.

Таким образом, обеспечивается гибкость при выборе и применении различных анализаторов. В зависимости от задач анализа проекта, может быть выбран определенный набор анализирующих модулей, при этом исчезает необходимость обеспечения коммуникации таких модулей напрямую. Такое связывание носит наиболее общий характер, что позволяет отразить зависимости компонентов в полном и универсальном виде, что впоследствии может быть использовано для более полного уточняющего анализа либо для интеграции в иные средства разработки программного обеспечения.

Для проверки данного метода на практике, был разработан прототип анализатора, использующий заложенную в анализируемую технологию семантическую информацию. Такой технологией (а вернее, таким набором технологий) стало сочетание языка разметки HTML и скриптового языка JavaScript. В качестве примера программного проекта было выбрано небольшое веб-приложение, представляющее собой интерактивную веб-страницу с многоэтапным процессом регистрации [4].

Полученная структура семантического узла, а также используемые атрибуты, типы компонентов и семантические роли показаны на рис. 1.

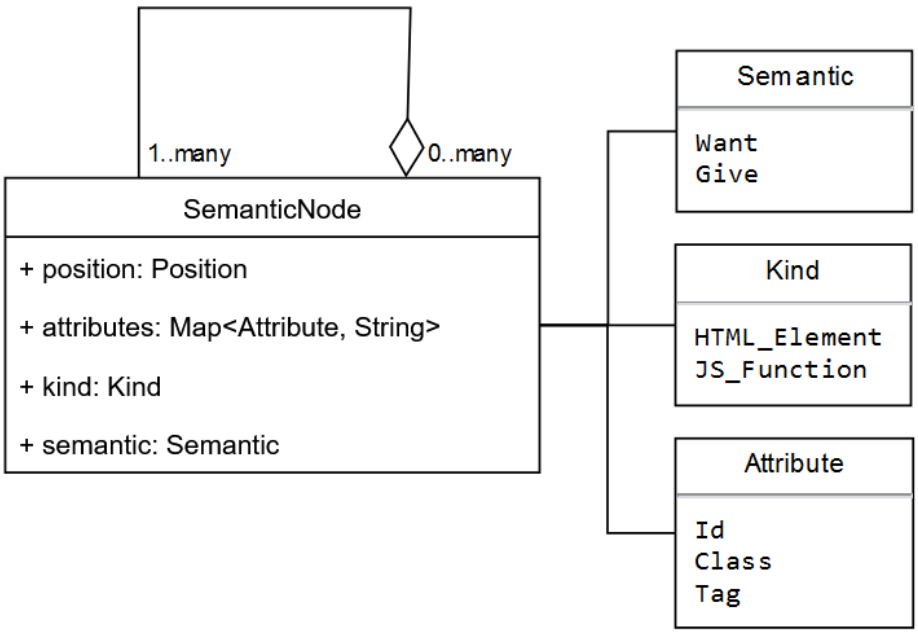

Рис. 1. Структура семантического узла, отражающая тип анализа

Типы анализируемых компонентов представлены двумя видами — тег HTML и функция JavaScript. Атрибуты, присутствующие на компоненте выявляются исходя из типа компонента: если это тег HTML, то атрибуты напрямую берутся из тега. Если же компонентом является функция JavaScript, то атрибуты берутся из её аргументов. Семантические роли Want и Give задают связь «использование», что позволяет сопоставить функции JavaScript с используемыми ими тегами HTML. Также, стоит заметить, что роли предполагают ассиметричную двустороннюю связь, что позволяет также обнаруживать теги HTML, которые используют определенные JavaScript функции в своих HTML атрибутах или теле тега.

На рис. 2 показан пример работы анализатора и фрагменты кода, сопоставленные при анализе. Связанные компоненты помечены синим цветом, а атрибуты, использованные для выявления связи, красным. В процессе связывания было получено отношение «использование», где использующим фрагментом является функция “getElementById” с параметром “nextBtn”, а используемым является тег “button”.

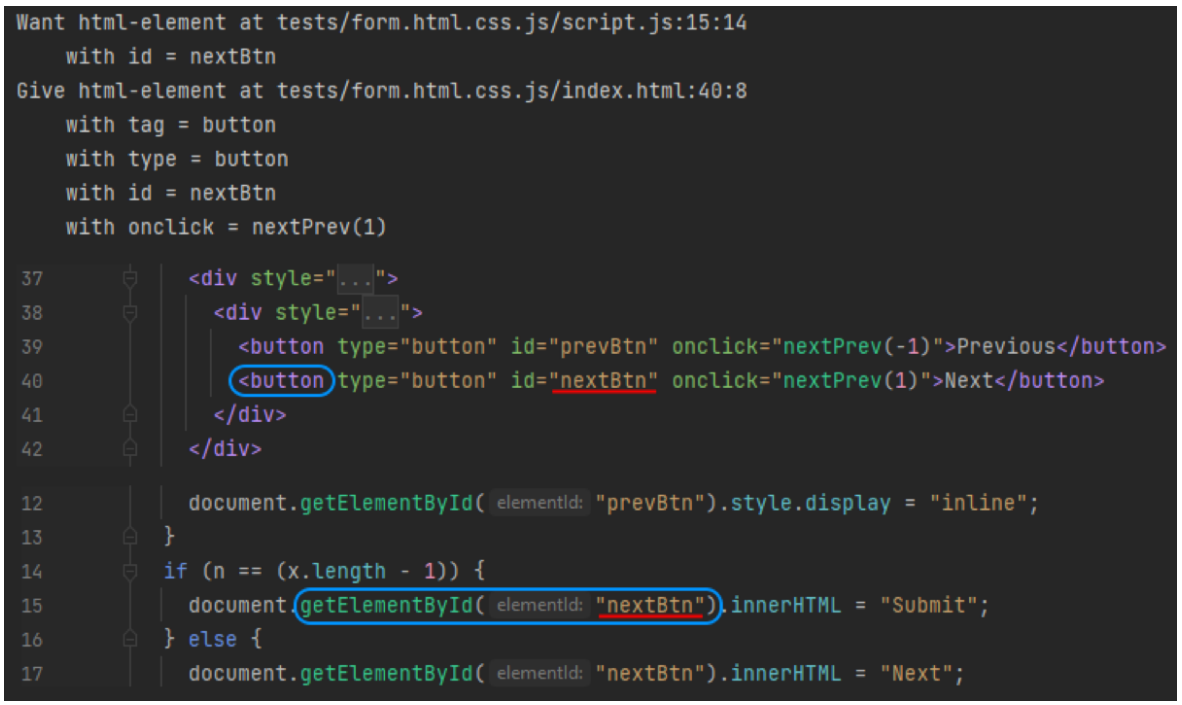

Рис. 2. Пример работы анализатора

В результате получен метод позволяющий извлекать семантическую информацию из исходного кода мультиязыковых программ, при этом стоит заметить, что для применения данного метода необходимо лишь предварительно сформировать антологию предметной области. В целом, метод имеет следующие характеристики:

1) семантическая информация отражена в виде антологии, что повышает гибкость анализатора и делает его независимым относительно анализируемых языков;
2) реализация метода может эффективно обрабатывать код путем распараллеливания процессов получения узлов и их связывания;
3) метод не полагается на определенный минимальный объем кода для проведения анализа, что позволяет успешно использовать его в средах разработки.

В качестве идей дальнейшего развития данного метода предполагаются следующие направления исследования:

- синтез единой онтологии для широкого множества популярных языков программирования (в первую очередь ОО языков);
- использование методов машинного обучения для облегчения синтеза онтологии программистом путем анализа множества возможных атрибутов, типов и семантик семантических узлов;
- исследование более широкого числа сценариев использования для интеграции метода.

1. T. van der Storm and J. J. Vinju, ‘‘Towards multilingual programming environments,’ Sci. Comput. Program. – 2015 – С. 143–149.
2. Главный репозиторий Multilingual Static Analysis tool – [Электронный ресурс]. – Режим доступа: https://github.com/MultilingualStaticAnalysis/MLSA (дата обращения: 21.02.2023).
3. CSE341: Programming Languages Winter 2013 – [Электронный ресурс]. – Режим доступа: https://courses.cs.washington.edu/courses/cse341/13wi/unit6notes.pdf (дата обращения: 13.04.2023).
4. Пример веб-приложения многоэтапной регистрации – [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/howto/howto_js_form_steps.asp (дата обращения: 15.03.2023).