БЕЗОПАСНОЕ ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ПРИ ИСПОЛЬЗОВАНИИ ORM

Наумова А.В.[1], Чукова Д.И.[2]

Данная работа посвящена рассмотрению актуальной проблемы разработки безопасного программного обеспечения при использовании ORM-подхода, а именно проблеме проектирования безопасной базы данных. Для решения этой проблемы предложено средство автоматизированного выявления потенциальных угроз. В статье рассмотрены особенности применения ORM, отмечено, что при использовании ORM-подхода задача проектирования базы данных сводится к задаче выделения классов в соответствии с принципами объектно-ориентированного программирования, отношений между ними и основных методов. Обосновано исключение возможности появления антипаттернов на этапе проектирования классов. Реализация подобного анализатора требует решения задачи выбора средств формализации баз данных и математического аппарата для идентификации потенциальных угроз. В статье приведен способ формализации баз данных с помощью дескрипционной логики ALCQI и использование регулярных выражений для поиска ситуаций, влияющих на уязвимость баз данных. Авторами рассмотрена семантика дескрипционной логики и составные концепты логики. Задав синтаксис и семантику дескрипционной логики, можно формализовать любую диаграмму классов. Таким образом, выразив антипаттерн в виде некоторого регулярного выражения на языке дескрипционной логики ALCQI, можно свести поиск шаблона к сопоставлению текстовой строки на наличие регулярного выражения.

Ключевые слова: *антипаттерны проектирования, дескрипционная логика, диаграммы классов, ALCQI, безопасная разработка, антишаблоны проектирования, формализация классов.*

DOI: *10.21681/2311-3456-2019-3-72-76*

Место базы данных в безопасности программного средства

Безопасность программного обеспечения складывается из многих компонент. Для создания безопасного продукта важно обеспечить защиту не только от сетевых атак или возможности утечки логинов пользователей, что делает возможным подбор пароля перебором, но и крайне важно спроектировать и реализовать надежное и отказоустойчивое решение, которое будет вести себя корректно и предсказуемо в любой ситуации. Правильно спроектированная база данных (БД) является важной составляющей надежного решения.

В настоящее время при разработке программных средств в роли архитекторов баз данных системы выступают сами разработчики. К сожалению, далеко не все разработчики обладают необходимыми для проектирования базы данных навыками. Такая ситуация приводит к снижению надежности всего программного средства.

Однако, существуют практики, использование которых освобождает от необходимости проектирования базы данных системы. Например, все чаще применяется ORM (Object Related Mapping) подход, используемый, например, в самом популярном фреймворке Django, предназначенном для написания веб-приложений на языке программирования Python. Принципы разработки безопасного программного средства с применением ORM описаны в статье [1].

Особенности применения ORM

ORM позволяет разработчикам не зависеть от особенностей устройства используемой системы управления базой данных (СУБД), работа с БД сводится к работе с данными объектно-ориентированных классов, то есть с классами и их экземплярами. Такой подход осуществляет преобразование данных классов в данные, пригодные для хранения в системе управления базой данных, и наоборот, пропадает необходимость написания SQL-кода для взаимодействия с СУБД.

Итак, согласно книге [2] и статьям [3, 4], в рамках использования ORM-подхода задача проектирования базы данных сводится к задаче выделения классов в соответствии с принципами объектно-ориентированного программирования, отношений между ними и основных методов. Операции над этими классами будут в дальнейшем с помощью средств ORM транслироваться на базу данных, именно поэтому крайне важно правильно выделить основные классы и методы, чтобы исключить возможность уязвимости БД.

Рассмотрим, как на этапе выделения классов обеспечить построение наиболее безопасного результата. Основными угрозами здесь является допущение так называемых антипаттернов, оказывающих негативное влияние на эффективность, корректность работы и отказоустойчивость системы.

Антипаттерны проектирование и их место в безопасности программного средства

Антипаттерны представляют собой часто употребляемые разработчиками шаблоны, применение которых имеет негативное влияние на работу программного средства. Примерами подобных антишаблонов являются такие приемы, как мягкое и жесткое кодирование, маги-

1 Наумова Александра Вячеславовна, разработчик, ЗАО «Перспективный мониторинг», г. Москва, Россия. E-mail: Aleksandra.Naumova@amonitoring.ru

2 Чукова Дарья Ильинична, старший преподаватель, РГУ нефти и газа (НИУ) имени И.М. Губкина, г. Москва, Россия. E-mail: chukova.d@gubkin.ru

ческая кнопка, спагетти код, множественное и ромбовидное наследование, зацикливание и многие другие случаи, рассмотренные в книге [5, 6] и статьях [7, 8]. Антипаттерны проектирования включают в себя не только само описание ошибки, но и последствия, которые она влечет за собой, и указание способов устранения ошибки (в общем случае это шаблон проектирования).

Для исключения возможности появления антипаттернов на этапе проектирования классов предлагается создать программное решение, которое бы автоматизировало процесс поиска ошибок. Такое средство позволит значительно сократить время проектирования и исключить человеческий фактор, приводящий к пропусканию ошибок.

Для реализации подобного средства автоматизации проверки наличия антипаттернов в классах и отношениях между ними необходимо решить ряд проблем:

- выбрать наиболее эффективный способ формализации для классов;
- определиться с математическим аппаратом, который бы позволял распознавать антипаттерны;
- разработать алгоритмы поиска антипаттернов, если для выбранного математического аппарата не существует нужных алгоритмов.

Формализация диаграмм классов

При рассмотрении диаграммы классов, отражающей общую структуру иерархии классов с учетом методов, полей и взаимосвязей между ними, на первый взгляд, кажется логичным использование теории графов для задачи поиска антишаблонов. Действительно, диаграмма классов – это ориентированный граф, где вершинами являются классы, а ребрами – связи между ними.

Однако, использование графов для формализации классов ограничивает круг антипаттернов, которые можно идентифицировать. Это связано с тем, что графы не позволяют формализовать поля и методы. Таким, образом применение теории графов для формализации классов подходит для случая, когда автоматизация поиска используется в основном для выявления структурных ошибок (например, поиск зацикливания, ромбовидного наследования и наличия несвязных графов).

Таким образом, необходимо найти способ формализации классов, наиболее полно отражающий все аспекты классов и отношений между ними. Из статьи [9] об онтологии в компьютерных системах заметим, что онтология это – детальная и всеобъемлющая формализация некоторой предметной области с помощью концептуальной схемы, чем и является схема базы данных.

В данном случае целесообразно применить описательную, или дескрипционную логику для целей формализации диаграммы классов. Дескрипционные логики широко используются для описания онтологий (например, язык веб-онтологий OWL-DL, который также упомянут в статье [9]) и разрабатывались в основном для этих целей.

Дескрипционные логики, подробное описание которых приведено в книге [10], оперируют понятиями «концепт» и «роль», соответствующими в других разделах математической логики понятиям «одноместный предикат» и «двуместный предикат». Интуитивно концепты используются для описания классов некоторых объектов, например, «Люди», «Машины». Роли используются для описания двуместных отношений между объектами, например, на множестве людей имеется двуместное отношение «X есть_родитель_для Y». С помощью языка дескрипционной логики можно формулировать утверждения общего вида — о классах вообще (всякая Машина имеется_в_собственности не более чем у одного Человека) и частного вида — о конкретных объектах (Иван имеет_в_собственности Тойоту).

Дескрипционная логика — это целое семейство языков представления знаний, описывающих понятия предметной области в недвусмысленном, формализованном виде. Логика ALC (Attributive Language with Complement) является одной из базовых и имеет два необходимых расширения Q и I [11]. Расширение Q позволяет определять качественные ограничения кардинальности ролей, а I – обратные роли. Данные расширения необходимы для формализации диаграммы классов, поэтому будет использоваться логика ALCQI.

Для того, чтобы задать какую-либо логику, необходимо задать ее синтаксис и семантику. Синтаксис описывает, какие выражения (концепты, роли, аксиомы и т.п.) считаются правильно построенными в данной логике. Семантика указывает, как интерпретировать эти выражения, то есть придает им формальный смысл.

Пусть $C_N = \{A1, \ldots, Am\}$ и $R_N = \{R1, \ldots, R_n\}$ — конечные непустые множества атомарных концептов и атомарных ролей (называемые также именами концептов и именами ролей). Опишем логику ALC. Пусть заданы непустые конечные множества атомарных концептов A и атомарных ролей R. Тогда составные концепты логики определяются следующим образом:

- выражения T (все множество) и $\perp$ (пустое множество) являются концептами;
- всякий атомарный концепт A является концептом;
- если C концепт, то его дополнение –C является концептом;
- если C концепт, а R роль, то выражения $\forall R.C$ и $\exists R.C$ тоже концепты.
- если C и D концепты, то их пересечение ($C \cap D$) и объединение ($C \cup D$) тоже являются концептами;

Аксиомой вложенности концептов называется выражение вида $C \subseteq D$. Аксиомой эквивалентности концептов — выражение вида $C \equiv D$, где C и D — произвольные концепты.

Семантика дескрипционной логики задается путем интерпретации ее атомарных концептов как множеств объектов, которые выбираются из фиксированного множества (домена), и атомарных ролей в виде множества пар объектов, т.е. бинарных отношений на домене.

Интерпретация $\mathcal{I}$ – это непустое множество ΔI (домен) и интерпретирующая функция. Эта функция сопоставляет каждому атомарному концепту A некоторое подмножество $AI \subseteq \Delta I$, а каждой атомарной роли R – некоторое подмножество $RI \subseteq \Delta I \times \Delta I$. Если пара индивидов принадлежит интерпретации некоторой роли R, то есть $(e,d) \in RI$, то говорят, что индивид d является R-последователем индивида e.

Расширение I отвечает за обратные роли: если R роль, то R^- – тоже является ролью, означающей обращение бинарного отношения. Интерпретация роли R^- – обратной R,

определяется аналогично интерпретации R, только пара индивидов меняется местами.

Отметим, что логику ALCQI можно перевести в логику предикатов первого порядка с двумя переменными. Данная логика разрешима, это позволяет переносить результаты о разрешимости, вычислительной сложности и разрешающих алгоритмах из области логики предикатов в область дескрипционных логик. Для диаграмм классов мы имеем дело только с TBox, так как объекты в явном виде в диаграммах классов не фигурируют.

Таким образом, задав синтаксис и семантику дескрипционной логики можно формализовать любую диаграмму классов. Основываясь на материалах статей [12-15], атрибут A типа K класса C представляется следующим образом:

$C \subseteq \forall A.K$

Операция f(R):P, задающая преобразование множества значений R в множество значений P, представляется ролью Pf(R), для которой справедливо выражение вида:

$C \subseteq \forall Pf(R).P \sqcap (\leq 1 Pf(R).\top)$

Отношение наследования между классами C1 и C2, представляется как:

$C2 \subseteq C1$, , где C1 — предок.

Подобным образом можно выразить и остальные типы связей между классами. Перейдем к вопросу о том, как осуществлять поиск антипаттернов.

Поиск антипаттернов

Каждый антипаттерн проектирования представляет собой некоторый шаблон. Представление диаграммы классов в виде набора утверждений дескрипционной логики представляет собой текстовую строку. Таким образом, выразив антишаблон в виде некоторого регулярного выражения на языке дескрипционной логики ALCQI, можно свести поиск шаблона к сопоставлению текстовой строки на наличие регулярного выражения.

Рассмотрим на примере антипаттерна «Множественное наследование» [16]. Множественное наследование заключается в том, что некоторый класс имеет несколько предков. Данная ошибка является критичной для языков программирования, не поддерживающих подобную структуру, например, для Java. Также при множественном наследовании может возникнуть перекрытие имен методов классов-предков [17]. Возникновение такой ситуации приводит к ошибке компилятора программы, так как он не позволяет определить, метод какого из классов-предков необходимо использовать. То есть, множественное наследование напрямую влияет на корректность работы программного средства, а, значит, на его безопасность.

Пусть имеется некоторая структура взаимодействия классов, содержащая класс Target_Class, имеющий два предка General_Class1 и General_Class2. На языке дескрипционной логики такая ситуация выражается следующим образом:

Target_Class $\subseteq$ General_Class1

Target_Class $\subseteq$ General_Class1

Для нахождения случая множественного наследования в общем случае для некоторого класса C необходимо проверить количество строк, кодирующих отношение наследования, то есть строк, соответствующих регулярному выражению:

C$\subseteq$ *, где * – любое значение.

И если количество строк, удовлетворяющих подобному регулярному выражению больше одного, то можно сделать вывод о наличии нескольких предков у класса C.

Аналогично, можно представить другие антипаттерны, допускаемые при проектировании классов, в виде регулярных выражений и реализовать алгоритмы [18-20] их нахождения в строке, кодирующей классы и их взаимодействие в виде набора утверждений дескрипционной логики ALCQI.

Заключение

В статье предложен метод автоматизации проверки схемы классов и взаимодействия между ними, являющихся отражением базы данных при использовании ORM-похода, на наличие антипаттернов проектирования. Такое средство автоматизации позволяет не только создавать более безопасное программное обеспечение, но и значительно экономить время, затрачиваемое на процесс проверки.

Приведен способ формализации классов с помощью дескрипционной логики ALC, предложено применение механизма регулярных выражений для выявления антипаттернов.

Для реализации средства автоматизации поиска антипаттернов в классах реализуются два основных модуля:

– модуль перевода исходных классов и отношений между ними в утверждения дескрипционной логики;
– модуль с методами поиска, реализующими алгоритмы поиска регулярных выражений в наборе утверждений дескрипцииионной логики.

Выбранный подход позволяет пополнять базу антипаттернов, так как способ формализации с помощью дескрипционной логики затрагивает все элементы классов, а любое регулярное выражение по сути является шаблоном и, следовательно, может отразить любой антипаттерн проектирования.

Литература

1. Токарчук А.М. Применение средств ORM для разработки безопасных веб-приложений // Безопасность информационных технологий, Том 17, Номер 1, 2010. С. 113-115.
2. T. Halpin. Object-Role Modeling Fundamentals: A Practical Guide to Data Modeling with ORM // Technics Publications LLC, 2015. 192 с. 3. Зверева О.М. Создание концептуальной модели данных в нотации ORM // Известия высших учебных заведений России, том 7, 2012. С. 105-112.
4. Oleynik P.P., Salibekyan S.M, Kuznetsov N.V. Implementation patterns of object static models for database applications: classical ORM-patterns and object-attribute approach // International journal of applied engineering research, том 10, номер 24, 2015. С. 41431-41443.
5. W. Brown, R. Malveau, H. McCormick III, T. Mowbray. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis // John Wiley & Sons, Inc., 1998. 156 с.
6. Colin J. Neill, Philip A. Laplante. Antipatterns: Identification, Refactoring, and Management // CRC Press, 2005. 336 с.

7. Змеев О.А., Иванова Л.С. Поиск артефактов проектирования. обзор подходов // Вестник томского государственного университета. Управление, вычислительная техника и информатика, номер 2, 2015. С. 81-90.
8. Neill, Colin J. Antipatterns in Systems Engineering: An Opening Trio // INCOSE International Symposium, том 22, номер 1, 2012. С. 1233—1245.
9. Guliakina N.A., Grakova N.V., Pivovarchik O.V., Fedotova A.V. Ontology-based design of intelligent systems for educational purposes // Открытые семантические технологии проектирования интеллектуальных систем. № 7. 2017. С. 239-244.
10. Кропотин А.А. Применение формализма дескрипционной логики для выявления семантических конфликтов концептуальных схем сущность связь // Интеллект. Инновации. Инвестиции. № 7. 2016. С. 93-98.
11. Майрин А.Ф. Разработка учебного приложения для проверки выполнимости баз знаний в дескрипционной логике ALC // Проблемы управления в социально-экономических и технических системах Сборник научных статей. 2018. С. 100-103.
12. D. Berardi, D, Calvanese, G. D. Giacomo. Reasoning on UML Class Diagram // Artificial Intelligence, том 168, 2005. С.70-118.
13. M. Sergievskiy. Description Logic Application for UML Class Diagrams Optimization // International Journal of Advanced Computer Science and Applications, том 8, номер 1, 2017. С. 268-272.
14. Сергиевский М.В., Конкин А.Ю. Использование дескрипционной логики для оптимизации диаграмм классов UML // Cloud of science, том 4, номер 3, 2017. С. 465 – 479.
15. Григорьев А.В., Гаркуша Н.А., Юсупова А.А. Интеллектуальный метод статического анализа кода на основе поиска шаблонов алгоритмов // Научно-технический вестник Поволжья. № 6. 2015. С. 115-118.
16. Трепаков И.С. Эффективная реализация таблиц виртуальных методов в языках с поддержкой ограниченного множественного наследования // МНСК-2018: Математика Материалы 56-й Международной научной студенческой конференции. 2018. С. 183.
17. Канатов А.В., Зуев Е.А. Концепция наследования в современных языках программирования // Труды Института системного программирования РАН. Т. 27. № 6. 2015. С. 169-188.
18. Брускин С.Н., Дружаев А.А., Марон А.И., Марон М.А. Эффективные методы построения алгоритмов поиска неисправностей в информационных системах // Интеллектуальные системы в производстве. Т. 15. № 3. 2017. С. 88-93.
19. Сычев О.А., Пахомов Д.А. Генерация описания регулярного выражения на естественном языке как инструмент помощи составителю регулярного выражения // Известия Волгоградского государственного технического университета. № 25 (152). 2014. С. 86-94.
20. Герасимов А.Н., Елизаров А.М., Липачев Е.К. Паттерны регулярных выражений в алгоритмах семантической обработки коллекций математических документов // Труды Казанской школы по компьютерной и когнитивной лингвистике TEL-2014. 2014. С. 43-45.

SAFE DATABASE DESIGN WHEN USING ORM

Naumova A.M.[3], Chukova D.I.[4]

This paper is concerned with the current problem of developing secure software using the ORM approach, specifically the problem of designing a secure database. To solve this problem, a tool for automated identification of potential threats is proposed. The authors discuss the ORM application aspects and note that when using the ORM approach, the database design task boils down to that of singling out classes as per the object-oriented programming principles, relationships between them and the main methods. The impossibility of anti-patterns appearing at the class design stage is substantiated. In order to implement such an analyzer, it is necessary to select database formalization tools and a mathematical apparatus to identify potential threats. The article suggests a way to formalize databases by means of the ALCQI description logic and using regular expressions to look for situations affecting database vulnerability. The authors examine descriptive logic semantics and logic composite concepts. By specifying the syntax and semantics of descriptive logic, one can formalize any class diagram. Thus, if an anti-pattern is expressed as some regular expression in the ALCQI descriptive logic language, the search for a pattern can be reduced to checking a text string for containing a regular expression.

Keywords: *design antipatterns, description logic, class diagrams, ALCQI, safe development, design anti-patterns, class formalization*

References

1. Tokarchuk A.M. Primeneniye sredstv ORM dlya razrabotki bezopasnykh veb-prilozheniy // Bezopasnost' informatsionnykh tekhnologiy, Tom 17, Nomer 1, 2010. S. 113-115.
2. T. Halpin. Object-Role Modeling Fundamentals: A Practical Guide to Data Modeling with ORM // Technics Publications LLC, 2015. 192 c.
3. Zvereva O.M. Sozdaniye kontseptual'noy modeli dannykh v notatsii ORM // Izvestiya vysshikh uchebnykh zavedeniy Rossii, tom 7, 2012. S. 105-112.
4. Oleynik P.P., Salibekyan S.M, Kuznetsov N.V. Implementation patterns of object static models for database applications: classical ORM-patterns and object-attribute approach // International journal of applied engineering research, том 10, номер 24, 2015. С. 41431-41443.
5. W. Brown, R. Malveau, H. McCormick III, T. Mowbray. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis // John Wiley & Sons, Inc., 1998. 156 c.
6. Colin J. Neill, Philip A. Laplante. Antipatterns: Identification, Refactoring, and Management // CRC Press, 2005. 336 c.
7. Zmeyev O.A., Ivanova L.S. Poisk artefaktov proyektirovaniya. obzor podkhodov // Vestnik tomskogo gosudarstvennogo universiteta. Upravleniye, vychislitel'naya tekhnika i informatika, nomer 2, 2015. S. 81-90.
8. Neill, Colin J. Antipatterns in Systems Engineering: An Opening Trio // INCOSE International Symposium, том 22, номер 1, 2012.

3 Naumova Alexandra Vyacheslavovna, developer, « Advanced Monitoring», Moscow, Aleksandra.Naumova@amonitoring.ru

4 Daria Chukova, Senior lecturer, Gubkin Russian State University of Oil and Gas (National Research University), Moscow, chukova.d@gubkin.ru

С. 1233—1245.
9. Guliakina N.A., Grakova N.V., Pivovarchik O.V., Fedotova A.V. Ontology-based design of intelligent systems for educational purposes // Otkrytyye semanticheskiye tekhnologii proyektirovaniya intellektual'nykh sistem. № 7. 2017. S. 239-244.
10. Kropotin A.A. Primeneniye formalizma deskriptsionnoy logiki dlya vyyavleniya semanticheskikh konfliktov kontseptual'nykh skhem sushchnost' svyaz' // Intellekt. Innovatsii. Investitsii. № 7. 2016. S. 93-98.
11. Mayrin A.F. Razrabotka uchebnogo prilozheniya dlya proverki vypolnimosti baz znaniy v deskriptsionnoy logike ALC // Problemy upravleniya v sotsial'no-ekonomicheskikh i tekhnicheskikh sistemakh Sbornik nauchnykh statey. 2018. S. 100-103.
12. D. Berardi, D, Calvanese, G. D. Giacomo. Reasoning on UML Class Diagram // Artificial Intelligence, том 168, 2005. С.70-118.
13. M. Sergievskiy. Description Logic Application for UML Class Diagrams Optimization // International Journal of Advanced Computer Science and Applications, том 8, номер 1, 2017. С. 268-272.
14. Sergiyevskiy M.V., Konkin A.YU. Ispol'zovaniye deskriptsionnoy logiki dlya optimizatsii diagramm klassov UML // Cloud of science, tom 4, nomer 3, 2017. S. 465 – 479.
15. Grigor'yev A.V., Garkusha N.A., Yusupova A.A. Intellektual'nyy metod staticheskogo analiza koda na osnove poiska shablonov algoritmov // Nauchno-tekhnicheskiy vestnik Povolzh'ya. № 6. 2015. S. 115-118.
16. Trepakov I.S. Effektivnaya realizatsiya tablits virtual'nykh metodov v yazykakh s podderzhkoy ogranichennogo mnozhestvennogo nasledovaniya // MNSK-2018: Matematika Materialy 56-y Mezhdunarodnoy nauchnoy studencheskoy konferentsii. 2018. S. 183.
17. Kanatov A.V., Zuyev Ye.A. Kontseptsiya nasledovaniya v sovremennykh yazykakh programmirovaniya // Trudy Instituta sistemnogo programmirovaniya RAN. T. 27. № 6. 2015. S. 169-188.
18. Bruskin S.N., Druzhayev A.A., Maron A.I., Maron M.A. Effektivnyye metody postroyeniya algoritmov poiska neispravnostey v informatsionnykh sistemakh // Intellektual'nyye sistemy v proizvodstve. T. 15. № 3. 2017. S. 88-93.
19. Sychev O.A., Pakhomov D.A. Generatsiya opisaniya regulyarnogo vyrazheniya na yestestvennom yazyke kak instrument pomoshchi sostavitelyu regulyarnogo vyrazheniya // Izvestiya Volgogradskogo gosudarstvennogo tekhnicheskogo universiteta. № 25 (152). 2014. S. 86-94.
20. Gerasimov A.N., Yelizarov A.M., Lipachev Ye.K. Patterny regulyarnykh vyrazheniy v algoritmakh semanticheskoy obrabotki kollektsiy matematicheskikh dokumentov // Trudy Kazanskoy shkoly po komp'yuternoy i kognitivnoy lingvistike TEL-2014. 2014. S. 43-45.