УДК 004.415.5
DOI: 10.18523/2617-3808.2022.5.26-30

Франків О. О.

ВИКОРИСТАННЯ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ДЛЯ ВІЗУАЛІЗАЦІЇ АРХІТЕКТУР ПРОГРАМНИХ МОДУЛІВ

У статті описано створений програмний комплекс ADAR для статичного аналізу програмного коду з подальшим створенням тривимірної моделі для візуалізації засобами доповненої реальності. Комплекс має природне для людини подання, що дає змогу на інтуїтивному рівні ефективно аналізувати складність зв'язків між різними частинами програмного коду, які виступають критерієм визначення зв'язності, зв'язаності або крихкості компонента.

Ключові слова: доповнена реальність, ARKit, статичний аналіз, архітектура ПЗ, мобільні пристрої.

Вступ

На сьогодні активного розвитку набув новий вид користувацького інтерфейсу на базі технологій доповненої реальності. Доповнена реальність – це технологія створення користувацького досвіду, в якому реальне середовище сприйняття користувача доповнюється віртуальними об'єктами (найчастіше візуальними та аудіальними) за допомогою програмних засобів.

Завдяки значним оптимізаціям як на програмному, так і на апаратному рівні, ця технологія стала придатною для використання на мобільних пристроях, а отже доступною для значної кількості користувачів. Провідні компанії, що займаються розробленням портативних приладів, упродовж останніх років активно працюють над створенням нових пристроїв введення-виведення інформації через доповнення реальності.

Спосіб взаємодії користувача з комп'ютером за допомогою саме доповненої реальності є не лише максимально інтуїтивним, а й більш безпечним порівняно з віртуальною реальністю, де користувач може втрачати координацію або відчувати неприємні симптоми через дезорієнтацію в просторі.

Водночас розроблення програмного забезпечення сьогодні тісно пов'язане з бізнесом, що зумовлює значну необхідність у мінімізації ресурсів, зокрема часових, витрачених на розроблення та впровадження. Важливою ознакою якісного програмного продукту є не лише відповідність визначеним технічним характеристикам, а й підтримуваність продукту. Ключову роль у цьому питанні відіграє правильність побудови архітектури конкретного модуля. Зважаючи на високу ціну помилки, допущеної на етапі проєктування, важливим є не лише розуміння архітектури модуля всіма учасниками команди, а й здатність швидко і вчасно провести діагностику коду для локалізації та ліквідації вад на якомога більш ранньому етапі. Створений комплекс ADAR завдяки швидкості роботи і простоті використання дає змогу виконувати таку діагностику так часто, як це необхідно.

Загальний принцип роботи комплексу

Програмний комплекс ADAR (Architecture Displayer in Augmented Reality) складається з двох програм – програми-аналізатора для macOS та програми-візуалізатора для iOS. У поточній версії існує можливість працювати з будь-якими проєктами, написаними мовою програмування Swift.

Процес роботи з ADAR можна умовно поділити на два етапи – аналіз та візуалізацію.

Програма-аналізатор отримує на вхід вихідний код програмного модуля, аналіз якого виконується. На виході користувач отримує тривимірну модель архітектури модуля у формі графа, що має найбільш придатне для сприйняття подання [4; 5]. Отримана модель записується програмою-аналізатором у файл.

Програма-візуалізатор – це мобільний застосунок, що може прочитати модель із файлу і правильно відобразити її в доповненій реальності. Окрім відображення застосунок також надає можливість взаємодіяти з моделлю.

Аналіз програмного коду

Аналізатор, після розбору коду, готує файл візуалізації. У поточній версії аналіз проводиться на основі набору абстрактних синтаксичних дерев, побудованих компілятором Swift [1]. Такий підхід усуває залежність від постійних змін синтаксису мови, яка дуже динамічно розвивається, в контексті парсингу вихідного коду.

Для створення моделі ми вибрали два види зв'язку між типами даних (структурами, класами, перерахуваннями) – property і staticFuncUsage. Тип зв'язку property вказує на залежність між екземплярами типу, а саме позначає використання екземпляру типу одного класу як властивості екземпляру іншого типу; натомість зв'язок staticFuncUsage вказує на зв'язок між типами, або ж між екземпляром типу та іншим типом. Для зручності ми також ввели додатковий тип зв'язку both, що позначає наявність обох згаданих зв'язків одночасно.

Попри порівняну простоту цього підходу, саме такі взаємні зв'язки, як «є частиною» та «використовує», допомагають робити судження про архітектуру модуля в цілому, хоча і не дають змоги претендувати на всеохопність отриманих результатів.

Тому природною моделлю каркаса архітектури став граф, вузлами якого є типи (в контексті Swift), а ребрами – зв'язки між ними. Оскільки за деяких умов окрім наявності того чи того зв'язку важливу роль може відігравати і кількість таких зв'язків, її можна передавати шляхом збільшення ваги ребра і, відповідно, переходом до зваженого графа.

Отримана модель є досить складною для розуміння, зокрема через наявність зв'язків між типами, створеними користувачем, і типами, описаними в бібліотеках, зокрема системних. Цю проблему може бути вирішено відкиданням вузлів, що відповідають типам, які не було задекларовано у вихідному коді безпосередньо, і, як наслідок, ребер, пов'язаних із ними.

Насамкінець вузли, що не мають жодного ребра, несуть мало інформації в цьому контексті, оскільки наявність таких ізольованих частин програми точно не може бути індикатором надмірної зв'язності коду і, як наслідок, крихкості. Тому такі вузли теж можна не враховувати в моделі.

Усі описані вище оптимізації проводять за один прохід із метою покращення швидкодії програми.

Підготовка моделі до візуалізації

Наступним етапом роботи програми-аналізатора є розміщення графа в тривимірному просторі. Цей процес передбачає надання кожному вузлу певної точки – його координати в просторі.

Задача відображення графа не нова, цю темю досить ґрунтовно досліджено. Існує багато відомих алгоритмів для досягнення цієї мети. Очевидним є те, що графи, з якими проводиться робота, є досить великими. Граф зв'язків усередині промислового проєкту може налічувати від кількох сотень до кількох сотень тисяч вузлів. Це задає певні обмеження для використовуваних алгоритмів у плані швидкодії.

У загальному випадку вузли можуть розміщуватись у довільних точках. Однак метою візуалізації є не стільки факт відображення графа, скільки полегшення сприйняття. Тому важливим є питання вибору такого розміщення вузлів, щоб сприйняття було найкращим.

Дослідження сьогодні оперують поняттям *міри естетичності*, що визначає простоту сприйняття і розуміння візуалізації людиною. Чим більшою є естетичність, тим легшим є сприйняття.

Міру естетичності визначають за відповідністю відображення певним критеріям. Ми розглядали такі критерії естетичності:

1. Жодні вузли не накладаються.
2. Ребра не перетинаються або ж кількість перетинів має бути мінімальною.
3. Суміжні вузли розміщені поблизу.
4. Група суміжних вузлів формує кластер.
5. Вузли та ребра розміщено рівномірно в межах доступної візуалізації.

У двовимірному просторі таким критеріям відповідають планарні графи. Однак більшість графів, що моделюють архітектуру реальних програмних модулів, не будуть планарними. Окрім того, критерій планарності має сенс лише в двовимірному просторі й має мало змісту в тривимірному, оскільки невиконання умов планарності можна легко вирішити виходом у третій вимір зі збереженням необхідної міри естетичності.

На сьогодні найбільш використовуваними є три типи алгоритмів, що з різною успішністю дають змогу отримати естетичне відображення в просторі: силові, зниження розмірності й розміщення за ознаками.

Жертвуючи, можливо, кращою швидкодією, ми віддали перевагу саме силовим алгоритмам, оскільки в загальному випадку вони допомагають отримати відображення з найвищою мірою

естетичності [3], що цілком відповідає меті розроблюваного програмного комплексу.

Основою силового алгоритму є фізична модель, де вузли графа визначають як точкові заряди з однаковим (позитивним) зарядом, що відштовхуються за законом Кулона; а ребра – як пружини, що з'єднують ці заряди і протидіють силі відштовхування, діючи за законом Гука. Суть алгоритму полягає у декларуванні такої фізичної моделі та ітеративному обрахунку стану системи (а саме позиції кожного вузла) через кожен момент часу, аж поки система не перейде в стан спокою.

Очевидно, слабкою стороною такого алгоритму є ітеративність. Більше того, умова зупинки обчислень є чіткою, однак кількість ітерацій неможливо точно порахувати наперед, хоча зрозуміло, що це число залежить від розміру Δt і чутливості системи, а саме, яке переміщення вузла буде достатньо мале, щоб можна було ним знехтувати і вважати, що система вже у спокої.

Окрім підбору найбільш оптимальних костант для алгоритму значну увагу ми також приділили оптимізаціям обходу графа. Зокрема, для зменшення часової складності одного обходу графа було застосовано модель Барнса–Хата, що використовується для наближеного обрахунку гравітаційної задачі N тіл [2].

Варто зауважити, що висока точність обрахунку координат для вузлів є надлишковою, адже головним завданням є максимізація міри естетичності. Та й висока точність обрахунків мало впливає на задоволення зазначеним критеріям естетичності. Але, жертвуючи точністю, можна значно покращити швидкодію програми.

Візуалізація моделі засобами доповненої реальності

Провівши опис підготовчої роботи, зосередимося на висвітленні реалізації етапу візуалізації в доповненій реальності. В проєкті ADAR інструментом візуалізації є спеціально розроблений iOS застосунок ADAR, що дає змогу відкривати файли *.adar та представляти їхній вміст у доповненій реальності.

Застосунок є інструментом за замовчуванням для роботи з відповідними файлами. Це значно спрощує роботу з ними, забезпечуючи роботу з будь-якої програми, що спрощує обмін та зберігання, зокрема і на хмарному сховищі, наприклад iCloud.

Окрім цього, застосунок ADAR має й інші корисні функції.

Застосунок будує тривимірну модель графа, розташовуючи її у визначеній користувачем точці простору, та надає користувачу можливість керувати розміщенням моделі за допомогою афінних трансформацій, якими легко керувати завдяки простим жестам.

Застосунок підтримує можливість роботи кількох пристроїв одночасно, об'єднуючи їх в одну сесію доповненої реальності, що забезпечує одночасне перебування в сесії до п'яти користувачів. Це дає змогу членам невеликих команд одночасно працювати з моделлю.

Програмні засоби розроблення

Програму-аналізатор імплементовано у формі консольного застосунку для macOS, написаного мовою Swift. Оскільки робота аналізатора потребує наявності компілятора Swift і значних ресурсів для швидкої побудови графа, використання саме macOS є очевидним.

Залежність програми-аналізатора від сторонніх модулів ми зменшили до мінімуму, залишивши тільки наявні в системі фреймворки, серед яких Foundation, SceneKit, що використовується для моделювання тривимірного простору й роботи з векторами, та CoreGraphics – для оптимізацій в роботі з графікою. Завдяки цьому програма не потребує додаткового встановлення.

Програму-візуалізатор було імплементовано у вигляді iOS-застосунку. Рушієм для доповненої реальності було обрано ARKit, оскільки це системний фреймворк, що допомагає максимально ефективно використовувати апаратне забезпечення пристрою, як-от ширококутну камеру чи LiDAR-сканер.

Серед функцій ARKit, що використовуються в застосунку, особливо слід виділити People Occlusion і Multipeer Connectivity.

Функція обминання людей використовує системний модуль для комп'ютерного зору і в автоматичному режимі розпізнає контур людини в кадрі, а також заміряє відстань до неї (програмними або апаратними засобами залежно від моделі пристрою). На основі цих даних рушій за допомогою шейдерів графічного ко-процесора здатний за потреби обрізати зображення віртуальних об'єктів так, наче людина затуляє їх у кадрі.

Multipeer Connectivity дає змогу максимально ефективно використовувати апаратне забезпечення пристрою для швидкого обміну повідомленнями про зміни в сесії доповненої реальності (наявність об'єктів, їх розміщення, зміна позиції спостерігача тощо). За допомогою цієї

технології було імплементовано можливість створення багатокористувацької сесії.

Використання лише стандартних модулів системи в чистому вигляді не лише усуває зайві залежності, спрощуючи підтримку, а й забезпечує автоматичне покращення якості доповнення реальності з кожним наступним оновленням операційної системи.

Висновки

Результатом цього дослідження стала розробка програмного продукту, що дає змогу проводити аналіз та діагностику архітектури програмних модулів за допомогою засобів доповненої реальності.

Під час дослідження було здійснено спробу проаналізувати різні типи зв'язку між структурними елементами програмного коду. Перевагою запропонованого підходу є простота розширення можливостей. У майбутньому для ширшого аналізу досить легко розширити набір типів зв'язків, а також ознак потенційних вад проєктування для діагностики.

У роботі також було запропоновано спосіб інтуїтивного відображення каркаса архітектури програми у формі графу в тривимірному просторі, а також критерії естетичності цього відображення.

Насамкінець, програма-візуалізатор, створена спеціально для роботи зі згенерованою моделлю графа, допомагає легко взаємодіяти. В майбутньому, однак, можна сконцентруватись на виведенні метаданих стосовно конкретних структурних елементів, а також швидкому пошуку і позиціонуванні.

Список літератури

1. Apple Inc. Документація архітектури компілятора Swift [Електронний ресурс] / Apple Inc. – 2020. – Режим доступу: https://swift.org/swift- compiler/#compiler-architecture.
2. Barnes J. A hierarchical O (N log N) force-calculation algorithm / J. Barnes, P. Hut // *nature*. – 1986. – No. 324 (6096). – Pp. 446–449.
3. Fruchterman T. J. Graph Drawing by Force-Directed Placement / T. J. Fruchterman, E. M. Reingold // Software – Practice & Experience. – 1991. – No. 21. – Pp. 1129–1164.
4. Gansner R. Emden. Topological fisheye views for visualizingLarge graphs / R. Emden Gansner, K. Yehuda, S. North // Visualization and Computer Graphics, IEEE Transactions. – 2005. – No. 411. – Pp. 457–468.
5. Lespinats S. Visualization and exploration of high-dimen-sional data using a "force directed placement" method: application to the analysis of Genomic signatures [Electronic resource] / S. Lespinats, A. Giron, B. Fertil. – 2005. – Mode of access: https://www.semanticscholar.org/paper/Visualisation-and-exploration-of-high-dimensional-a-Lespinats-Giron/dbfab802232c362c0997a5656ff546c7d32d6a07.

References

Apple Inc. (2020). Документація архітектури компілятора Swift. https://swift.org/swift- compiler/#compiler-architecture.

Barnes, J., & Hut, P. (1986). A hierarchical O (N log N) force-calculation algorithm. *nature*, *324* (6096), 446–449.

Fruchterman, T. J., & Reingold, E. M. (1991). Graph Drawing by Force-Directed Placement. *Software – Practice & Experience*, *21*, 1129–1164.

Gansner, R. Emden, Yehuda, K., & North, S. (2005). Topological fisheye views for visualizing Large graphs. *Visualization and Computer Graphics, IEEE Transactions*, *411*, 457–468.

Lespinats, S., Giron, A., & Fertil, B. (2005). *Visualization and exploration of high-dimen-sional data using a "force directed placement" method: application to the analysis of Genomic signatures*. https://www.semanticscholar.org/paper/Visualisation-and-exploration-of-high-dimensional-a-Lespinats-Giron/dbfab802232c362c0997a5656ff546c7d32d6a07.

O. Frankiv

USING AUGMENTED REALITY FOR VISUALIZING ARCHITECTURES OF SOFTWARE MODULES

Nowadays the technology of augmented reality has become available for a wide audience of users because of a big number of software and hardware enhancements and optimizations done in the last years. The fact that the smartphone is a suitable and relatively cheap device having all the hardware required makes the technology even more accessible and thus widespread. Furthermore, the interaction with three-dimensional objects in space may have positive impact on user's perception of information. These both facts make the technology of augmented reality a good choice for displaying complex data.

The analysis of software plays a significant role in development as it is vital to keep the code clean and sustained all the time. Poor quality code may be unsustainable to the extent it must be fully replaced which results in big losses of resources. In terms of quality checks the analysis must be informative and consume as few resources as possible to be executed so that it is appropriate to perform it regularly. That is the reason for this process to be automated and made convenient to execute and percept.

The new system for automatic software analysis is described in this article. ADAR (Architecture Displayer in Augmented Reality) software is best suitable for code coupling and cohesion analysis as it uses three-dimensional graph to display connectivity between parts of software module. High coupling and low cohesion might inform the developers of severe architectural mistakes that may lead to high code fragility. With the use of AR technology the result of high coupling detection analysis in the form of graph is presented in augmented reality to provide user the information in a highly intuitive way.

This article also covers different approaches to graph visualization in three-dimensional space. The criteria that allow to achieve high level of aesthetics relative to this problem are stated in paper. The problem of using the force-directed algorithms in terms of high-aesthetic graph visualization is described in details and some arguments pro their usage are given.

Keywords: augmented reality, ARKit, static analysis, software architecture, mobile devices.

Матеріал надійшов 10 вересня 2022р.