УДК 004.624
DOI https://doi.org/10.32782/2663-5941/2025.3.2/18

Войтко В.В.
Вінницький національний технічний університет

Позур М.Ю.
Вінницький національний технічний університет

МЕТОД ДИНАМІЧНИХ ВИКЛИКІВ В .NET ШЛЯХОМ ГЕНЕРАЦІЇ IL КОДУ В ПРОЦЕСІ ВИКОНАННЯ ЗАСТОСУНКУ

У статті розглядається розробка методу здійснення динамічних викликів в .NET шляхом генерації IL коду в процесі виконання застосунку. Було проаналізовано основний спосіб здійснення динамічних викликів в .NET з використанням рефлексії. Визначено, що ключовим недоліком рефлексії є її швидкодія, тому існує необхідність пошуку альтернативного рішення для здійснення динамічних викликів. Було розроблено метод, який дозволить здійснювати динамічні виклики аналогічно до рефлексії, але з вищою швидкодією. Запропоноване рішення опирається на використання структури метаданих аналогічно до рефлексії, де метадані членів типу даних асоціюються з відповідними статичними викликами без використання динамічного виклику рефлексії. Така структура використовує асоціацію між власними метаданими типу та метаданими типу рефлексії. Реалізована з використанням хеш-таблиць, що дозволяє отримувати необхідні метадані об'єкту аналогічно до рефлексії. Генерація метаданих відбувається в процесі виконання застосунку через використання спеціально визначеного функціоналу або автоматично при спробі отримання метаданих типу, якщо останній відсутній у системі, що дозволяє використовувати запропонований метод як для типів даних, які були доданими в процесі виконання застосунку, так і анонімних типів даних. Статичні виклики, які асоціюються з відповідними метаданими, генеруються разом з метаданими. Для генерації статичних викликів пропонується використання функціоналу платформи .NET, що дозволяє розширити байт-код додатку в процесі виконання. До цієї категорії відноситься механізм Expression, що дозволяє будувати вирази в процесі виконання додатку та компілювати їх в лямбда-функції, та Reflection.Emit, що дозволяє написання програмних компонентів безпосередньо на рівні байт-коду в процесі виконання застосунку. Запропонований метод динамічних викликів було реалізовано у вигляді бібліотеки. Функціонал генерації статичних викликів розроблено з використанням механізму Expression. Швидкодія запропонованого методу перевірена з використанням бібліотеки Benchmark.NET.

Ключові слова: *метапрограмування, .NET, рефлексія, генерація коду, Expression.*

Постановка проблеми. Сучасне програмне забезпечення є доволі складним та масштабним, що потребує значних ресурсних і часових затрат з боку розробників. У зв'язку з цим популярності набуває використання засобів метапрограмування в процесі розробки програмного забезпечення. До таких засобів входять інструменти, що дозволяють генерацію вихідного коду, і механізми мов програмування та фреймворків, що дозволяють використовувати підходи метапрограмування в процесі виконання додатку [1, с. 33]. В контексті платформи .NET одним із найбільш поширених засобів метапрограмування є механізм рефлексії [2, с. 16]. Цей механізм дозволяє отримувати метадані додатку в процесі його виконання та виконувати низку операцій з цими метаданими. Одним із найбільш типових сценаріїв використання рефлексії в .NET є здійснення динамічних викликів [3, с. 10]. Під динамічним викликом тут розуміємо виклик, що здійснюється з використанням динамічного визначення імені (dynamic name resolution). Ключовою особливістю таких викликів є те, що член типу даних, для якого відбувається виклик, визначається не на етапі компіляції, а під час виконання застосунку. Найпоширенішим сценарієм використання такого типу викликів є серіалізація та десеріалізація даних. Проте, ключовим недоліком рефлексії є швидкодія викликів у порівнянні зі статичними викликами [4], що обумовлює необхідність пошуку альтернативних рішень.

Метою роботи є підвищення швидкодії процесу створення програмних засобів шляхом розробки та впровадження методу динамічних викликів для платформи .NET з генерацією IL коду при виконанні застосунку, що дозволить

покращити швидкодію програмного забезпечення в сценаріях, які опираються на використання динамічних викликів.

Об'єктом дослідження постають процеси розробки програмного забезпечення на платформі .NET з використанням метапрограмування.

Предметом дослідження є моделі та методи розробки програмного забезпечення на платформі .NET з метапрограмуванням.

Аналіз останніх досліджень і публікацій. У напрямку оптимізації динамічних викликів на платформі .NET є низка сучасних публікацій. У публікації [5] розглянуто використання механізму Expression для реалізації функціоналу рефлексії в контексті зчитування значення властивостей. Автор описує використання механізму динамічної компіляції Expression в лямбда функції для реалізації функціоналу динамічних викликів рефлексії. У результаті було отримано аналогічний до рефлексії функціонал зчитування властивостей, що має кращу швидкодію викликів у порівнянні з рефлексією. У публікації [6] розглянуто метод оптимізації рефлексії за рахунок використання динамічної компіляції Expression у лямбда функцію та кешування вже скомпільованої функції.

Таким чином, існуючі рішення опираються на динамічну генерацію статичних викликів та повторне використання згенерованого коду для досягнення кращих показників швидкодії. Проте, варто зазначити, що розглянуті рішення описують лише процес генерації та використання лямбда-функції, але не розробку механізму, що дозволив би зберігання таких функцій та їх повторне використання.

Постановка завдання. Головною задачею є розробка методу, який дозволить здійснення динамічних викликів у .NET за рахунок генерації байт-коду (IL коду) в процесі виконання застосунку. Такий метод опиратиметься на можливість платформи .NET динамічно розширювати власний байт-код у процесі виконання, що дозволяє динамічно генерувати лямбда-функції. Використання лямбда-функції в комбінації з метаданими дозволить отримати функціонал динамічних викликів, схожий до рефлексії, але з вищими показниками швидкодії.

Виклад основного матеріалу. При розробці власного механізму здійснення динамічних викликів у .NET, у першу чергу, варто розглянути рефлексію, яка є основним механізмом платформи .NET для здійснення такого типу викликів. Розглянемо виконання динамічного виклику за допомогою рефлексії (рис. 1).

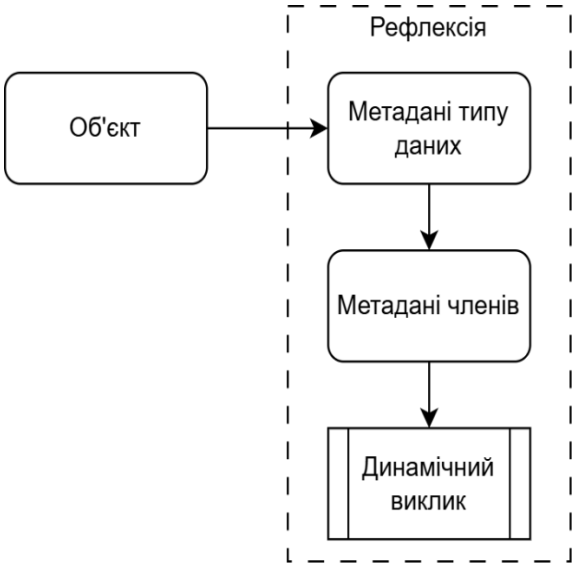

Рис. 1. Модель здійснення динамічного виклику в .NET з використанням рефлексії

Для здійснення динамічного виклику необхідно мати об'єкт, для члену якого відбуватиметься такий виклик. Далі потрібно отримати метадані типу даних (System.Type), що містить у собі метадані всіх членів відповідного типу даних. Для динамічного виклику того чи іншого члену необхідно отримати його метадані. Використовуючи об'єкт метаданих, виконується сам динамічний виклик. У публікації [4] розглядається здійснення такого виклику на рівні середовища виконання платформи .NET Common Language Runtime (CLR). Виконання динамічного виклику за допомогою рефлексії потребує великої кількості операцій на рівні CLR, що призводить до нижчої порівняно зі статичними викликами швидкодії.

Проте, якщо розглядати весь процес здійснення динамічного виклику з використанням рефлексії, то можна побачити, що визначення члену, для якого здійснюється виклик, відбувається на етапі отримання метаданих цього члену, а не на етапі здійснення самого виклику. Якщо асоціювати об'єкт метаданих члену зі статичним викликом цього члена, можна отримати аналогічний до рефлексії функціонал, але з вищою швидкодією.

Проте з таким підходом виникають деякі проблеми:

– необхідність створення власної системи метаданих, адже неможливо додати посилання на статичні виклики до існуючих метаданих;

– необхідність генерації відповідних статичних викликів.

Ключовою задачею власної системи метаданих є забезпечення швидкого доступу до метаданих. Для отримання кращих показників швидкодії у порівняні з рефлексією розроблена система метаданих має забезпечувати кращу або аналогічну швидкість знаходження необхідних метаданих. Для цього доцільно використовувати структуру метаданих, що є схожою до рефлексії. Визначимо основні елементи такої системи:

– *ExpReflection*. Статичний клас, що зберігає у собі посилання на загальну структуру даних та надає доступ до функціоналу розроблюваної системи.

– *ExpRefelctionTypeData*. Клас, що відповідає за метадані типу даних у розроблюваній системі.

– *ExpReflectionPropertyData*. Клас, що відповідає за метадані властивості типу даних у розроблюваній системі.

Така структура (рис. 2) дозволить відтворити елементи рефлексії, що є важливими в контексті здійснення динамічних викликів, та забезпечити аналогічну швидкість знаходження метаданих.

Поле *data* статичного класу *ExpReflection* містить посилання на хеш-таблицю, яка використовується для асоціації метаданих типу рефлексії з метаданими типу розробленої системи. Ключем такої хеш-таблиці є об'єкт метаданих типу рефлексії (*System.Type*), а не назва типу даних. Це пов'язано з обраховуванням хешу для посилальних типів даних у CLR. Значення хешу кожного об'єкта обраховується один раз та зберігається в заголовку цього об'єкта [7]. Таким чином, обрахунок хешу для посилального типу даних зводиться до повернення вже обрахованого результату. Оскільки метадані в CLR існують в єдиному екземплярі, то кожен виклик *object.GetType*() повертатиме об'єкт *System.Type*, що матиме вже обрахований хеш. Так можна досягти кращих показників швидкодії пошуку метаданих типу в розробленій системі.

Об'єкт, що відповідає за метадані типу (*ExpReflectionTypeData*) в розробленій системі має дві структури даних, що містять метадані членів цього типу. Одна структура – зв'язний список, інша – хеш-таблиця. Список використовується, коли необхідно отримати метадані всіх членів, тоді як хеш-таблиця – лише у випадках, коли відбувається пошук конкретного члену з використанням його імені.

Розроблена система розглядається в контексті динамічних викликів саме властивостей класів (зчитування та присвоєння їх значень). Це пов'язано з тим, що такі виклики найчастіше використовуються саме для властивостей. Отже, структура розглядає об'єкт метаданих властивості класу (*ExpReflectionPropertyData*). Такий об'єкт має посилання на відповідні функції, що здійснюють статичний виклик зчитування (*getValueFunc*) або присвоєння (*setValueFunc*) значення цієї властивості для певного об'єкту відповідного типу даних.

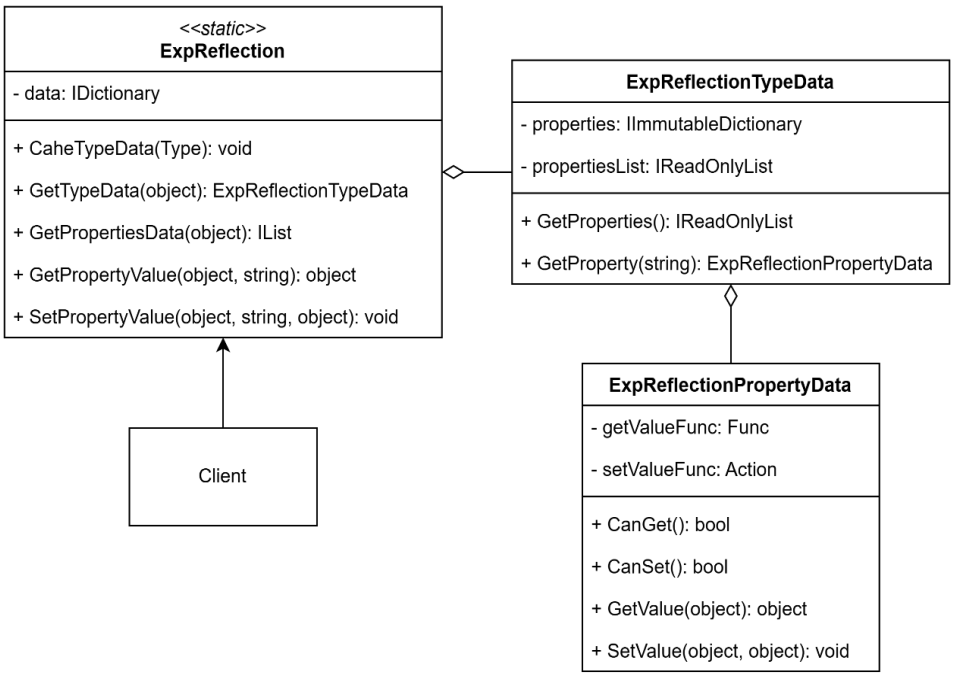

Рис. 2. UML-діаграма розроблюваної системи для здійснення динамічних викликів

Динамічний виклик з використанням розробленої системи описується моделлю (рис. 3).

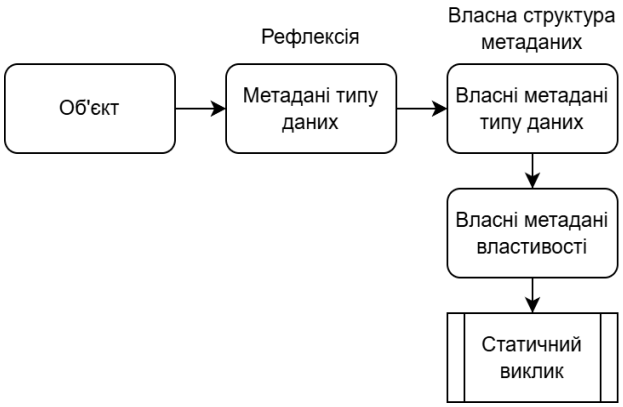

Рис. 3. Модель здійснення динамічного виклику в .NET з використанням розробленої системи

З моделі здійснення динамічного виклику та структури розробленої системи видно, що метадані властивості мають асоціюватися з відповідними статичними викликами. Проте такі статичні виклики необхідно згенерувати перед тим, як їх використовувати. Для цього можна використовувати як механізм Expression, так і Reflection.Emit. Обидва підходи дозволять згенерувати лямбда-функцію, що виконуватиме відповідний статичний виклик у процесі виконання застосунку. Такі функції повинні генеруватися разом із додаванням метаданих до системи.

Оскільки розроблена система метаданих використовується лише для здійснення динамічних викликів, то нема необхідності відразу генерувати всі метадані. Найбільш доцільною тут буде генерація метаданих лише для типів, для яких здійснюються динамічні виклики. Для цього процес отримання метаданих типу з такої системи має враховувати той факт, що тип може бути відсутнім (рис. 4).

Таким чином, якщо метадані необхідного типу відсутні в системі, то відбудеться їх генерація. Така генерація включає в себе не лише створення відповідних об’єктів, а й генерацію статичних викликів для властивостей цього типу даних.

Отже, для реалізації запропонованого методу динамічних викликів потрібно:

1) визначити власну структуру метаданих, що складається з метаданих членів типу, метаданих типів та статичного класу, що надає доступ до цих метаданих;

2) метадані членів мають містити посилання на відповідні статичні виклики;

Рис. 4. Блок-схема алгоритму процесу отримання метаданих властивостей типу даних

3) визначити операцію генерації метаданих типу у розробленій системі, яка повинна забезпечити генерацію як об’єктів метаданих, так і відповідних статичних викликів для членів цього типу;

4) при отриманні метаданих типу необхідно перевіряти, чи є такий тип у системі, якщо ні, то генерувати його метадані, використовуючи операцію, визначену кроком 3;

5) для здійснення динамічних викликів використовувати розроблену систему метаданих.

Запропонований метод динамічних викликів порівнювався за швидкодією з рефлексією. Для цього було використано бібліотеку Benchmark.NET [8]. Порівняння відбувалося як для зчитування значення властивості, так і для присвоєння. Оскільки процес отримання метаданих є частиною здійснення динамічного виклику, то його було включено до порівняння. Результати порівняння швидкодії розробленого методу динамічного виклику та швидкодії динамічного виклику з використанням рефлексії наведено у таблиці 1.

За результатами порівняння можна зробити висновок, що запропонований метод здійснення динамічних викликів є в середньому на 60% швидшим у порівняні з використанням рефлексії. Варто зазначити, що запропонована реалізація методу використовує механізм Expression для генерації лямбда-функцій у процесі виконання додатку.

Таблиця 1

Порівняльний аналіз здійснення динамічних викликів присвоєння та зчитування значень властивості з використанням розробленого методу динамічного виклику та рефлексії

Method	Mean	Error	StdDev	Ratio
ExpReflectionGet	15.79 ns	0.643 ns	0.602 ns	0.41
ReflectionGet	38.50 ns	1.182 ns	1.106 ns	1.00
ExpReflectionSet	16.22 ns	0.783 ns	0.733 ns	0.38
ReflectionSet	42.21 ns	0.279 ns	0.146 ns	1.00

Висновки. Розроблено метод здійснення динамічних викликів у .NET за рахунок генерації IL коду в процесі виконання застосунку. Розроблений метод опирається на використання окремої системи метаданих, де метадані кожного члену типу даних асоціюються з відповідними статичними викликами. Для генерації таких викликів використовується механізм Expression платформи .NET. Швидкодія розробленого методу є в середньому на 60% вищою за швидкодію рефлексії в сценаріях здійснення динамічних викликів зчитування та присвоєння значень властивостей. Однією з переваг запропонованого методу є те, що генерація метаданих та статичних викликів відбувається в процесі виконання застосунку, що дозволяє його використання для типів даних, які були динамічно додані, та для анонімних типів даних.

Список літератури:

1. Bock J., Hazzard K. Metaprogramming In . NET. Manning Publications Co. LLC, 2012.
2. Ingebrigtsen E. Metaprogramming in C#: Automate Your .NET Development and Simplify Overcomplicated Code. Packt Publishing, Limited, 2023.
3. Beaumont A., Bakhtiari Bastaki B. An Investigation into the Prevalence of Reflection Techniques in Distributed Microsoft .Net NuGet Artefacts. *ICSCA 2022: 2022 11th International Conference on Software and Computer Applications*, м. Melaka Malaysia. New York, NY, USA, 2022. URL: https://doi.org/10.1145/3524304.3524329.
4. Warren M. Why is reflection slow?. URL: https://surl.li/kxttwl (дата звернення: 17.05.2025).
5. Haytam Z. C# Expression Trees: Property Getters. URL: https://blog.zhaytam.com/2020/11/17/expression-trees-property-getter/ (дата звернення: 17.05.2025).
6. Ricardo Peres. Using Generated Methods Instead of Reflection. URL: https://weblogs.asp.net/ricardoperes/using-generated-methods-instead-of-reflection (дата звернення: 17.05.2025).
7. GetHashCode inside CLR: Reference types. URL: https://tearth.dev/posts/gethashcode-inside-clr-reference-types/ (дата звернення: 17.05.2025).
8. BenchmarkDotNet. URL: https://benchmarkdotnet.org/index.html (дата звернення: 17.05.2025).

Voitko V.V., Pozur M.Yu. METHOD OF EXECUTING DYNAMIC CALLS IN .NET USING IL GENERATION IN RUNTIME

The article discusses the development of a method for making dynamic calls in .NET by generating IL code during application execution. The main method for making dynamic calls in .NET which relies on using reflection was analyzed. It was determined that the key disadvantage of the reflection in a context of making dynamic calls is its performance, so there is a need to find an alternative solution for making dynamic calls. A method was developed that will allow making dynamic calls in a same way as by using reflection, but with a better performance. The proposed solution is based on the use of a metadata structure, similar to the reflection, where the metadata of type members are associated with corresponding static calls. This way using a reflection dynamic call can be avoided. Such a structure defines an association between the own type metadata and the reflection type metadata, implemented using hash tables. This allows obtaining the necessary metadata for the object in a same way as by using reflection. Metadata generation occurs during application execution through the use of specially defined functionality or automatically when attempting to obtain type metadata if the latter is missing in the system, which allows using the proposed method for both data types that were added during application execution and anonymous data types. Static calls associated with the corresponding metadata are generated along with the metadata. To generate static calls, it is proposed to use the functionality of the .NET platform, which allows for extending the application's bytecode

in runtime. This category includes the Expression mechanism, which allows you to build expressions during application execution and compile them into lambda functions, and Reflection.Emit, which allows you to write program components directly at the bytecode level during application execution. The proposed method of dynamic calls was implemented as a library. The functionality of generating static calls was developed using the Expression mechanism. The performance of the proposed method was tested using the Benchmark.NET library.

Key words: *metaprogramming, .NET, reflection, code generation, Expression.*