УДК 004.42:004.051 DOI https://doi.org/10.35546/kntu2078-4481.2025.3.2.29

Н. О. КОМЛЕВА
кандидат технічних наук, доцент,
завідувач кафедри інженерії програмного забезпечення
Національний університет «Одеська політехніка»
ORCID: 0000-0001-9627-8530

# МЕТОД ТРАНСФОРМАЦІЇ ФОРМАЛІЗОВАНИХ ВИМОГ У РОЗШИРЕНІ UML-МОДЕЛІ НА ОСНОВІ ГРАМАТИЧНОГО ПІДХОДУ

*У даній статті розглядається актуальне завдання трансформації формалізованих вимог у розширені UML-моделі, яке має важливе значення для підвищення ефективності процесів проєктування інтелектуальних та критичних систем. Дослідження зосереджується на поєднанні граматичного підходу з механізмами UML-профілю, що дозволяє забезпечити автоматизовану перевірку консистентності вимог, відображення властивостей невизначеності та імовірнісних характеристик, а також підвищити точність і адаптивність створюваних моделей. Актуальність теми зумовлена зростанням складності сучасних інформаційних систем, де вимоги часто залишаються нечіткими, суперечливими чи змінюваними, що ускладнює їх формалізацію та подальше відображення у традиційному UML.*

*Запропоновано методологію, що базується на дворівневій граматиці: атрибутна граматика використовується для узгодження вимог та побудови їх узгодженої множини, а трансформаційна граматика визначає відображення цих вимог у UML-елементи відповідно до їх атрибутів (confidence, likelihood, context, priority, resources). У статті наведено формальні правила трансформації для функціональних, нефункціональних, вимог до даних та сценарних вимог, які реалізуються через створення варіантів використання, класів, обмежень, діаграм послідовностей та діаграм станів. Особлива увага приділяється механізму призначення стереотипів, що зберігають семантику вихідних специфікацій, а також трасуванню, яке забезпечує прозорий зв'язок між вимогами та елементами UML-моделі.*

*Практичну придатність методу продемонстровано на прикладі інтелектуальної медичної системи моніторингу пацієнтів. Проведене дослідження засвідчило низку переваг розробленого підходу: скорочення часу на актуалізацію моделей, підвищення рівня узгодженості між вимогами та їхніми модельними представленнями, можливість відображення нечітких і ймовірнісних характеристик. Порівняння з класичним UML без профілю показало, що запропонований метод надає ширші можливості для моделювання адаптивних систем у середовищах з високим рівнем невизначеності. Разом з тим визначено обмеження: потреба у налаштуванні граматики під конкретний предметний домен і необхідність інтеграції з CASE-засобами для автоматизації трансформацій.*

*Подальші дослідження передбачають кілька напрямів розвитку методології: автоматизована генерація UML-діаграм у сучасних CASE-засобах; а також інтеграція підходу з SysML v2 як перспективного стандарту системної інженерії.*

***Ключові слова:*** *UML, вимоги, граматика, трансформація, інженерія програмного забезпечення, інтелектуальні системи, моделювання, профіль UML.*

N. O. KOMLEVA
PhD, Associate Professor,
Head of Software Engineering Department
Odesa Polytechnic National University
ORCID: 0000-0001-9627-8530

# GRAMMAR-BASED METHOD FOR TRANSFORMING FORMALIZED REQUIREMENTS INTO EXTENDED UML MODELS

*This article addresses the relevant task of transforming formalized requirements into extended UML models, which is of great importance for increasing the efficiency of designing intelligent and critical systems. The study focuses on combining a grammatical approach with UML profile mechanisms, which makes it possible to ensure automated consistency checking of requirements, representation of uncertainty and probabilistic properties, and improvement of the accuracy and adaptability of the created models. The relevance of the topic is determined by the increasing complexity of modern information systems, where requirements often remain vague, contradictory, or changing, which complicates their formalization and further representation in traditional UML.*

*A methodology is proposed based on a two-level grammar: attribute grammar is used for requirements reconciliation and for constructing the their consistent set, while transformation grammar defines the mapping of these requirements*

*into UML elements according to their attributes (confidence, likelihood, context, priority, resources). The article provides formal transformation rules for functional, non-functional, data, and scenario requirements, which are implemented through the creation of use cases, classes, constraints, sequence diagrams, and state diagrams. Special attention is paid to the mechanism of assigning stereotypes that preserve the semantics of the original specifications, as well as to tracing, which ensures transparent links between requirements and UML model elements.*

*The practical applicability of the method is demonstrated using the example of an intelligent medical patient monitoring system. The study revealed several advantages of the developed approach: reduction of the time required for updating models, improvement of the level of consistency between requirements and their model representations, and the ability to represent fuzzy and probabilistic properties. A comparison with classical UML without a profile showed that the proposed method provides broader opportunities for modeling adaptive systems in environments with a high level of uncertainty. At the same time, certain limitations were identified: the need to customize grammar for a specific application domain and the requirement for integration with CASE tools to automate transformations.*

*Further research involves several directions for the development of the methodology: automated generation of UML diagrams in modern CASE tools; as well as integration of the approach with SysML v2 as a promising system engineering standard.*

***Key words:*** *UML, requirements, grammar, transformation, software engineering, intelligent systems, modeling, UML profile.*

**Постановка проблеми**

У сучасних програмних системах усе більшої актуальності набуває проблема формалізації та моделювання вимог, які часто є нечіткими, ймовірнісними та змінюваними залежно від контексту. Традиційна нотація UML, орієнтована на стабільні та однозначні специфікації, виявляється недостатньою для повноцінного відображення таких характеристик. Це призводить до зростання ризику конфліктів між вимогами, ускладнює процес їхньої перевірки та збільшує витрати на підтримку моделей у динамічних середовищах.

Класичні методи інженерії вимог у подібних умовах демонструють обмежену ефективність, оскільки не враховують властивості невизначеності, багатоваріантності та залежності від зовнішніх факторів. Як наслідок, розробники змушені підтримувати паралельну документацію або вносити численні ручні корективи у модельні представлення, що негативно впливає на їхню цілісність та актуальність.

Для подолання цих обмежень доцільним є поєднання методів граматичного аналізу, які забезпечують формальне узгодження вимог, із розширеними механізмами UML-профілю, здатними відобразити нечіткість, ймовірнісність та контекстну залежність. Такий інтегрований підхід створює умови для автоматизованої трансформації формалізованих вимог у UML-моделі з розширеними стереотипами та тегованими значеннями.

**Аналіз останніх досліджень і публікацій**

Сучасна практика моделювання програмних систем демонструє широкий спектр підходів до розширення можливостей UML. Одним із поширених напрямів є створення UML-профілів для специфічних предметних областей. Такі профілі дають змогу розширювати стандартний апарат UML за допомогою стереотипів, тегованих значень та обмежень, що забезпечує його адаптацію до нетипових завдань [1, с. 27; 2, с. 87]. Наприклад, існують профілі для моделювання просторових даних, нечітких траєкторій або адаптивного програмного забезпечення [3, с. 2503; 4, с. 1720]. Вони демонструють гнучкість UML як метамови, проте здебільшого орієнтовані на окремі класи систем і не забезпечують комплексної підтримки невизначених та змінюваних вимог.

Окремий науковий напрям зосереджений на методах роботи з нечіткими та ймовірнісними вимогами, де застосовуються апарати штучного інтелекту, нечіткої логіки та методи машинного навчання. Зокрема, пропонуються підходи до пріоритизації вимог у випадку невизначеності, валідації часових обмежень, виражених природною мовою, або побудови самоадаптивних систем [5, с. 1; 6, с. 115]. Такі рішення дозволяють формалізувати властивості вимог, але вони здебільшого залишаються відокремленими від UML-моделювання та не інтегруються у його нотацію [7, с. 1905].

Третій напрям становлять формальні граматики та трансформаційні системи у моделюванні. Використання контекстно-вільних, атрибутних та графових граматик надає засоби для формалізації природномовних специфікацій, перевірки сумісності моделей та здійснення їх автоматичних перетворень [8, с. 1; 9, с. 185826]. Наприклад, граматики застосовуються для перевірки моделей трансляції, для побудови графових перетворень UML-діаграм або для синтезу семантичних дій у процесі аналізу вимог [10, с. 153]. Ці дослідження підкреслюють ефективність граматичного апарату у завданнях формалізації, проте залишаються обмеженими на рівні окремих операцій аналізу чи трансляції.

Таким чином, в оглянутих роботах простежується активний розвиток трьох паралельних напрямів – розширення UML за допомогою профілів, використання методів нечіткої логіки для формалізації вимог, а також застосування формальних граматик для перевірки та трансформації моделей. Проте відсутній інтегрований підхід, який поєднує формальну перевірку вимог на основі граматик із їх подальшим автоматизованим відображенням у UML-профіль, здатний фіксувати невизначеність та змінюваність.

**Формулювання мети дослідження**

Метою роботи є розробка методу трансформації формалізованих та узгоджених вимог у розширені UML-моделі на основі граматичного підходу, який забезпечує збереження властивостей невизначеності, імовірнісності та контекстної змінюваності у процесі моделювання складних адаптивних систем.

Для досягнення мети в роботі вирішуються такі задачі:

1) аналіз сучасних методів інженерії вимог та засобів моделювання на основі UML, включаючи профілі для специфічних предметних областей;

2) формалізація вимог із використанням атрибутної граматики для узгодження та побудови множини вимог, придатної до подальшої трансформації;

3) розробка системи трансформаційних правил, що відображають формалізовані вимоги у структурні та поведінкові елементи розширеної UML-моделі;

4) демонстрація застосування методу на прикладі критичної предметної області (інтелектуальна медична система) та оцінка його практичної придатності;

5) визначення переваг, обмежень та перспектив розвитку підходу, зокрема його інтеграції з CASE-засобами.

**Викладення основного матеріалу дослідження**

**Формалізація вимог та їхнє модельне відображення**

Трансформація вимог у UML-моделі потребує попередньої їхньої формалізації та узгодження, а також визначення апарату для подання властивостей невизначеності й змінюваності у межах нотації UML.

Формалізація здійснюється за допомогою контекстно-вільної та атрибутної граматики, що поєднує структурні правила побудови вимог із набором семантичних характеристик. Контекстно-вільна частина забезпечує однозначне синтаксичне подання, тоді як атрибутна дозволяє зберігати додаткові параметри, важливі для подальшого аналізу.

Основними атрибутами є: *confidence* – ступінь довіри до вимоги, *likelihood* – імовірність активації, *context* – умови виконання, *priority* – відносна важливість, *resources* (ρ) – оцінка необхідних витрат.

На основі цих параметрів здійснюється багаторівнева перевірка узгодженості: логічна, функціональна, ресурсна та контекстна. Результатом є узгоджена множина вимог $R^*$, яка зберігає інформацію про всі ключові характеристики та слугує підґрунтям для подальшої трансформації у модельне подання.

Щоб забезпечити відображення властивостей невизначеності у UML, використовується спеціалізований профіль, побудований на розширенні стандартної метамоделі. Він включає сутності *Requirement*, *Uncertainty Source*, *Variability Model* та їхні зв'язки з UML-артефактами.

У межах профілю передбачено низку стереотипів: <<fuzzyRequirement>> – нечіткі вимоги, <<probabilistic>> – ймовірнісні, <<variableRequirement>> – змінювані; <<temporalVariant>> – залежні від часу; <<evolvableRequirement>> – такі, що еволюціонують у процесі життєвого циклу.

Ці стереотипи супроводжуються тегованими значеннями (*confidence*, *trustDecayRate*, *likelihood*, *context*, *priority*, *resources*), які дозволяють відобразити кількісні та якісні характеристики вимог безпосередньо у UML-моделях. Додатково визначено інваріанти та обмеження, що забезпечують автоматизовану перевірку консистентності та підтримують трасованість між вимогами і модельними елементами.

Таким чином, формалізовані й узгоджені вимоги з множини $R^*$ можуть бути безпосередньо трансформовані у UML-моделі, доповнені профілем для роботи з недосконалими та змінюваними специфікаціями.

**Дворівнева граматика трансформації вимог у UML**

Розроблений підхід ґрунтується на поєднанні двох граматичних рівнів, які працюють послідовно:

1. Рівень $G_A$ – атрибутна граматика узгодження вимог. Вона забезпечує синтаксичний і семантичний аналіз початкових специфікацій, формує абстрактне синтаксичне дерево (AST), перевіряє їхню узгодженість за атрибутами та повертає узгоджену множину $R^*$.

2. Рівень $G_T$ – трансформаційна граматика UML-відображень. Цей рівень одержує на вхід множину $R^*$, що містить вже перевірені вимоги, і застосовує правила трансформації до вузлів AST, утворюючи елементи UML, доповнені профілем.

Дворівнева грамматика подається як кортеж:

$$G = \langle G_A, G_T, R^*, \tau \rangle, \quad (1)$$

де $G_A$ – атрибутна граматика узгодження, $R^*$ – множина узгоджених вимог, що є виходом $G_A$, $G_T$ – граматика трансформації вимог у UML, $\tau : R^* \rightarrow \text{UML}^+$ – відображення множини узгоджених вимог у елементи UML з профілем.

Ця дворівнева грамматика забезпечує перехід від неструктурованих вимог до повноцінної UML-моделі з урахуванням невизначеності та варіативності.

Трансформаційна граматика $G_T$ визначається у вигляді п'ятірки:

$$G_T = \langle N, \Sigma, P, S, A_T \rangle,$$

де $N$ – множина нетерміналів, що позначають абстрактні класи вимог (функціональні, нефункціональні, дані, сценарії); $\Sigma$ – множина терміналів, які відповідають конкретним умовам та діям; $P$ – набір правил трансформації;

$S$ – початковий символ, що позначає окрему вимогу з множини $R^*$; $A_T$ – система атрибутів, узгоджена з тегованими значеннями UML-профілю.

Кожне правило трансформації подається у вигляді:

$$T = (LHS, RHS, C), \tag{2}$$

де $LHS \subseteq AST$ – фрагмент дерева розбору, що відповідає вимозі або композиції вимог; $RHS$ – UML-конструкція (UseCase, Class, Component, Constraint, Activity, State); $C$ – умови застосовності правила, які залежать від атрибутів вимоги.

Відображення трансформації визначається як:

$$\tau : R^* \to UML^+, \tag{3}$$

де $UML^+$ – UML, розширений спеціалізованим профілем.

Для збереження властивостей невизначеності введемо множину стереотипів:

$$S = \{\langle \text{fuzzyRequirement} \rangle, \langle \text{probabilistic} \rangle, \langle \text{variableRequirement} \rangle\}.$$

Функція призначення стереотипів:

$$\sigma : R^* \to S \cup \varnothing, \tag{4}$$

де для кожної вимоги $r_i$:

– якщо $\text{conf}(r_i) < 1$, тоді $\sigma(r_i) = \langle \text{fuzzyRequirement} \rangle$;
– якщо $\text{likelihood}(r_i) \in (0,1)$, тоді $\sigma(r_i) = \langle \text{probabilistic} \rangle$;
– якщо $\text{context}(r_i)$ змінюється динамічно, тоді $\sigma(r_i) = \langle \text{variableRequirement} \rangle$;
– інакше – стереотип не застосовується.

Таким чином, кожна вимога отримує додаткову семантичну мітку, що переноситься у модель.

Розглянемо загальні правила трансформації для різних вимог та приклади їх застосування.

*Функціональні вимоги.* Загальне правило:

$$Func\,\text{Re}quirement(\varphi, \psi, A) \xrightarrow{C(A)} UseCase(\psi)^{\sigma(A)} + Class(\psi Controller)^{\sigma(A)},$$

де умова φ стає логічною умовою (умова переходу), дія ψ – назвою варіанту використання, а атрибути зберігаються як теговані значення.

Приклад 1 (медична система).

Requirement (φ = «Temp > 38», ψ = «NotifyDoctor», $A$ = {priority = 0.8, confidence = 0.9}) → *Use Case* «NotifyDoctor» зі стереотипом <<fuzzyRequirement>> та клас *NotificationController* з методом +notifyDoctor().

Приклад 2 (банківська система).

Requirement (φ = «Balance < 0», ψ = «BlockAccount», $A$ = {likelihood = 0.95, priority = 1.0}) → *Use Case* «BlockAccount» зі стереотипом <<probabilistic>> та клас *AccountController* з методом +blockAccount().

Приклад 3 (система бронювання квитків).

Requirement (φ = «SeatsAvailable > 0», ψ = «BookTicket», $A$ = {priority = 0.9, context = OnlinePortal}) → *Use Case* «BookTicket» без додаткового стереотипу, але з тегами *priority* = 0.9 і *context* = *OnlinePortal*; створюється клас *BookingController* з операцією + bookTicket().

*Нефункціональні вимоги*. Загальне правило:

$$NonFunctional\,\text{Re}q(\text{Pr}edicate, A) \xrightarrow{C(A)} Constra\,\text{int}(\text{Pr}edicate)^{\sigma(A)}.$$

Приклад 1 (електронна комерція).

NonFunctionalReq(«ResponseTime < 2 s»,{confidence = 0.85}) → обмеження на клас *WebServer* зі стереотипом <<fuzzyRequirement>>.

Приклад 2 (сховище даних).

NonFunctionalReq(«Availabilit > 0.999»,{likelihood = 0.98, priority = 0.9}) → обмеження на компонент *StorageCluster* зі стереотипом <<probabilistic>>.

Приклад 3 (мобільний застосунок).

NonFunctionalReq(«BatteryUsage < 5 %/hour», {context = Mobile, priority = 0.8}) → обмеження на клас *AppCore* зі стереотипом <<variableRequirement>>, що вказує на залежність від контексту виконання.

*Вимоги до даних*. Загальне правило:

$$Data\,\text{Re}q(Metric(Element), A) \xrightarrow{C(A)} Class(Element) + Constra\,\text{int}(Metric)^{\sigma(A)}.$$

Приклад 1 (медична база даних).

DataReq(«Completeness(PatientRecord) > 0.95»,{context = Hospital,confidence = 0.9}) → клас *PatientRecord* з тегом *completeness* = 0.95 та стереотипом <<fuzzyRequirement>>.

Приклад 2 (система ідентифікації).

DataReq(«Accuracy(FaceRecognitionData)>0.98»,{priority=1.0,context=Security}) → клас *FaceRecognitionData* з обмеженням *accuracy* = 0.98 та стереотипом <<variableRequirement>>.

Приклад 3 (система електронних платежів).

DataReq(«Consistency(TransactionLog) = Strong»,{likelihood = 0.95}) → клас *TransactionLog* з тегом *consistency* = *Strong* і стереотипом <<probabilistic>>, що вказує на ймовірнісну гарантію консистентності.

*Сценарні вимоги*. Загальне правило:

$$Scenario\left(\{ri \to rj\}, A\right) \xrightarrow{C(A)} UML\left(r_i, r_j\right)^{\sigma(A)}\Big),$$

де $\{r_i \to r_j\}$ – залежність між вимогами, що виражає послідовність дій, а UMLDiagram може бути діаграмою послідовностей, діяльності або станів.

Приклад 1 (авторизація користувача, діаграма послідовностей).

Scenario(«EnterCredentials → ValidateCredentials → AccessGranted») → діаграма послідовностей надає повідомлення *enterCredentials*() → *validateCredentials*() → *grantAccess*(). Умова переходу: [*passwordCorrect*].

Приклад 2 (система моніторингу, діаграма станів).

Scenario(«Monitoring → AlarmTriggered») з умовою [$SpO_2 < 90$] → діаграма станів виконує перехід зі стану *Monitoring* до стану *AlarmTriggered* з «умовою переходу» [$SpO_2 < 90$].

Приклад 3 (онлайн-магазин, діаграма діяльності).

Scenario(«AddToCart → Checkout → Payment → Confirmation») → діаграма діяльності містить послідовність дій із розгалуженням:

– [paymentSuccessful] → дія «SendConfirmation»;
– [paymentFailed] → дія «NotifyUser».

На виході функції трансформації τ формується UML-модель:

$$M = \tau(R^*) = \{E_i \mid \exists\, r_j \in R^* : T(r_j) = E_i\}, \tag{5}$$

де *M* містить структурні (*класи, компоненти*), поведінкові (*варіанти використання, діаграми станів, активностей, послідовностей*) та обмежувальні (*OCL, Constraints*) елементи. Усі вони доповнені стереотипами та тегами, що відображають семантику вимог.

Таким чином, трансформація не лише створює UML-діаграми, а й переносить у них всю інформацію про невизначеність, пріоритетність і ресурси, збережену на рівні граматичної моделі.

Задля забезпечення прозорості змін використовується відображення:

$$\text{Trace} : R^* \leftrightarrow \text{UML}^+, \tag{6}$$

де кожній вимозі відповідає один або кілька UML-елементів.

Це дозволяє відстежувати вплив зміни вимоги на модель, контролювати повноту покриття вимог у моделі та підтримувати консистентність між рівнем специфікацій і UML.

**Інтегрована методологія трансформації вимог у UML-моделі**

Запропонована методологія забезпечує повний цикл переходу від початкових специфікацій до розширених UML-моделей і ґрунтується на дворівневій граматиці. Вона поєднує механізми узгодження вимог та їх подальшої трансформації у модельні представлення.

*Етап 1. Формалізація вимог*. На першому етапі неформальні специфікації подаються у вигляді продукцій атрибутної граматики $G_A$, що входить до складу дворівневої системи (1). У результаті будується абстрактне синтаксичне дерево, у вузлах якого фіксуються семантичні характеристики вимог (атрибути *confidence*, *likelihood*, *context*, *priority*, *resources*). Це створює основу для подальшої перевірки узгодженості. Обмеженням цього етапу є залежність від повноти словника граматики: нечіткі формулювання можуть бути інтерпретовані неоднозначно.

*Етап 2. Перевірка узгодженості*. Далі здійснюється контроль отриманих вимог за логічними, функціональними, ресурсними та контекстними критеріями. На виході формується узгоджена множина $R^*$, яка використовується як вхід для трансформаційної граматики $G_T$ у (1). Якщо вимога не проходить одну з перевірок, вона відсікається або маркується як непридатна для відображення у UML.

*Етап 3. Підготовка до трансформації*. Кожна узгоджена вимога з множини $R^*$ описується у вигляді фрагмента AST, який може бути використаний у правилах трансформації. Як визначено у (2), кожне правило описується трійкою (LHS,RHS,C), де ліва частина відповідає піддереву вимоги, права частина задає UML-конструкцію, а умова застосовності залежить від атрибутів.

*Етап 4. Відображення у UML*. Трансформація множини вимог визначається функцією (3). Для кожної вимоги застосовується функція призначення стереотипів σ (4), яка зіставляє значення атрибутів з профілем UML. Наприклад, вимоги з неповною впевненістю отримують стереотип <<fuzzyRequirement>>, імовірнісні – <<probabilistic>>, а залежні від динамічного контексту – <<variableRequirement>>. Теговані значення переносяться у модель автоматично, що забезпечує збереження семантики вимог. Обмеженням цього етапу є можливість конфлікту між кількома стереотипами, який розв'язується за допомогою політики пріоритетності.

*Етап 5. Побудова UML-діаграм*. У результаті трансформації формується множина UML-елементів M, визначена у (5). Вона включає структурні моделі (класи, компоненти), поведінкові діаграми (варіанти використання, діяльності, послідовності, стани), а також обмежувальні елементи (OCL-висловлювання та UML-Constraints). Логічні умови з вимог перетворюються на умови переходів, які у UML позначаються у квадратних дужках. Автоматична генерація поведінкових моделей можлива лише для сценаріїв з повним порядком виконання; у випадку часткового порядку потрібне додаткове уточнення вручну.

*Етап 6. Трасування та валідація*. Для збереження відповідності між вимогами та елементами моделі використовується відображення (6). Це дозволяє відстежувати походження UML-елементів, контролювати покриття вимог та виявляти пропуски. На завершальному етапі застосовуються інваріанти UML-профілю, які перевіряють допустимість значень атрибутів, правильність призначення стереотипів і відсутність конфліктних позначень.

Таким чином, інтегрована методологія забезпечує послідовний перехід:

$$Req \xrightarrow{G_A} R^* \xrightarrow{\tau} UML^+,$$

де кожен етап має формальне обґрунтування, що гарантує повну узгодженість процесу трансформації.

**Приклад застосування інтегрованої методології трансформації вимог у UML-моделі**

Для демонстрації методології розглянемо предметну область інтелектуальної медичної системи моніторингу пацієнтів. У таких системах критично важливими є швидкість реакції, точність інтерпретації показників та відстежуваність вимог у моделі.

Початкова множина вимог $Req = \{R_1, R_2, R_3\}$:

– $R_1$ (функціональна): якщо показник $SpO_2 < 90$, система повинна ініціювати тривогу;
– $R_2$ (нефункціональна): час реакції системи не повинен перевищувати 2 секунд;
– $R_3$ (сценарна): після успішної авторизації лікар має отримати доступ до історії пацієнта протягом 1 хвилини.

За допомогою атрибутної граматики $G_A$ та перевірки узгодженості було підтверджено коректність усіх вимог та отримано узгоджену множину $R^*$.

Визначимо трансформації для різних типів UML-діаграм.

*Діаграма варіантів використання:*

– вимога $R_1$ трансформується у варіант використання «TriggerAlarm» зі стереотипом <<probabilistic>>;
– вимога $R_3$ породжує варіант використання «AccessHistory» зі стереотипом <<variableRequirement>>.

*Діаграма класів:*

– для $R_1$ створюється клас *AlarmController* з операцією +triggerAlarm();
– для $R_3$ створюється клас *HistoryController* з операцією +accessHistory();
– для $R_2$ у класі *System* фіксується обмеження {ResponseTime < 2 s}, позначене стереотипом <<fuzzyRequirement>>.

*Діаграма послідовностей:*

– вимога $R_3$ породжує сценарій: login() → validateUser() → accessHistory();
– умова переходу: [within 1 minute].

*Діаграма станів:*

– вимога $R_1$ відображається як перехід *Monitoring* → *AlarmTriggered* з умовою [$SpO_2 < 90$];
– вимога $R_3$ доповнює модель станами *LoggedOut* → *LoggedIn* → *HistoryAccessed* із часовим обмеженням «≤ 1 хвилина».

Узгоджені вимоги та їх відображення в UML-елементах подано нижче (табл. 1).

Таким чином, наведений приклад демонструє повний цикл трансформації від формалізованих вимог до узгоджених UML-моделей зі стереотипами та трасувальними зв'язками, що підтверджує практичну придатність запропонованої методології.

**Оцінювання ефективності методу трансформації формалізованих вимог**

Запропонований метод трансформації формалізованих вимог у розширені UML-моделі продемонстрував низку переваг у процесі проєктування.

По-перше, було досягнуто суттєвого скорочення часу на актуалізацію моделей. Завдяки автоматизованому застосуванню трансформаційних правил оновлення UML-діаграм після зміни вимог відбувається значно швидше, ніж у випадку ручного редагування. Це особливо важливо для систем, що функціонують у динамічному середовищі та потребують регулярної адаптації.

Таблиця 1

**Трасування вимог до елементів UML-моделі**

| Вимога | UML-елементи | Стереотипи/теги | Типи діаграм |
|---|---|---|---|
| $R_1$: TriggerAlarm при $SpO_2 < 90$ | Use Case «TriggerAlarm»; Class AlarmController; State «Monitoring → AlarmTriggered» | <<probabilistic>>, likelihood = 0.95, priority = 0.9 | Use Case, Class, State |
| $R_2$: ResponseTime < 2 s | Constraint на клас System | <<fuzzyRequirement>>, confidence = 0.85 | Class (з обмеженням) |
| $R_3$: Login → AccessHistory ≤ 1 хв | Use Case «AccessHistory»; Class HistoryController; Sequence «login → validate → access»; State «LoggedIn → HistoryAccessed» | <<variableRequirement>>, context = Hospital | Use Case, Class, Sequence, State |

По-друге, методологія зберігає властивості невизначеності та варіативності вимог. Використання атрибутів у поєднанні зі стереотипами профілю UML дозволяє безпосередньо відобразити нечіткі, імовірнісні або контекстно залежні специфікації у модельних елементах. Таким чином, UML-моделі перестають бути лише детермінованими схемами, а відображають реальні обмеження та умови функціонування системи.

По-третє, у процес було інтегровано автоматизовану перевірку консистентності, що знижує ризик суперечностей і забезпечує вищу надійність кінцевої моделі. Узгоджена множина вимог $R^*$ гарантує, що до етапу трансформації потрапляють лише ті специфікації, які пройшли логічний, функціональний, ресурсний і контекстний контроль.

При порівнянні з класичним UML-моделюванням (без використання розширеного профілю) виявлено суттєву відмінність. Стандартний UML дозволяє будувати структурні та поведінкові діаграми, однак він не надає засобів для формалізованого відображення невизначеності, імовірнісних властивостей або контекстної мінливості. У запропонованій методології ці характеристики зберігаються й інтегруються безпосередньо у модель, що робить її більш адекватною для опису сучасних складних систем.

Разом із тим метод має певні обмеження. Насамперед він потребує налаштування граматики під конкретний предметний домен, оскільки словник продукцій і правила трансформацій залежать від термінології та типових сценаріїв системи. Крім того, для повноцінного застосування необхідна інтеграція з CASE-засобами, здатними підтримувати розширений UML-профіль і забезпечувати автоматизоване трасування.

## Висновки

У роботі представлено метод трансформації узгоджених вимог у розширені UML-моделі, що базується на поєднанні граматичного підходу з механізмами UML-профілю. Запропонована методологія забезпечує збереження семантичних характеристик вимог, підтримує властивості невизначеності, імовірнісності та контекстної змінюваності, а також гарантує узгодженість між специфікаціями та модельними елементами. Поєднання атрибутного граматичного аналізу та розширеного UML-профілю створює комплексний інструмент для моделювання адаптивних систем. Це дозволяє скоротити час актуалізації моделей, зменшити ризик суперечностей і підвищити точність відображення предметної області в умовах динамічного середовища.

Подальші дослідження передбачають розвиток методології у двох ключових напрямах: автоматизована генерація UML-діаграм у сучасних CASE-засобах та розширення підходу для моделювання поведінкових сценаріїв. Окремо слід відзначити перспективу інтеграції з SysML v2, яка може стати основою для застосування методології у сфері системної інженерії.

## Список використаної літератури

1. Allian A.P., Nakagawa E.Y., Martinez J., Oliveira Jr. E., et al. Variability Implementation and UML-Based Software Product Lines. UML-Based Software Product Line Engineering with SMarty. Springer, Cham, 2022. С. 27–40. DOI: 10.1007/978-3-031-18556-4_2

2. Cu C., Ye X., Zheng Y. Xlinemapper: a product line feature-architecture-implementation mapping toolset. 41st International Conference on Software Engineering: Companion Proceedings (ICSE ’19). IEEE Press, Piscataway, 2019. С. 87–90. DOI: 10.1109/ICSE-Companion.2019.0004516

3. Abdelmadjid L. Uncertain Decision-Making Requirements Formalizing with CF UML model. Procedia Computer Science. 2022. Vol. 192. С. 2503–2512. DOI: 10.1016/j.procs.2021.12.247

4. Cazzola W., Olivares-Corichi I. Bridging the model-to-code abstraction gap with fuzzy logic: FLiRTS 2 for UML class diagrams. Software and Systems Modeling. 2022. Vol. 21. С. 1717–1742. DOI: 10.1007/s10270-021-00899-6

5. Cardiel-Ortega J. J., López-Robles J. R., Otegi-Olaso J. R., Gamboa-Rosales H. Probabilistic Fuzzy System for Evaluation and Failure Modes in FMEA. Processes. 2024. Vol. 12, Is. 6. Article № 1197. DOI: 10.3390/pr12061197

6. Dalpiaz F., Ferrari A., Franch X., Palomares C. Natural Language Processing for Requirements Engineering: The Best Is Yet to Come. IEEE Software. 2018. Vol. 35, No. 5. С. 115–119. DOI: 10.1109/MS.2018.3571242

7. Yang M., Ban A. Automated UML Class Diagram Generation from Textual Requirements Using NLP Techniques. JOIV International Journal on Informatics Visualization. 2024. Vol. 8, No. 3–2. С. 1905–1915. DOI: 10.62527/joiv.8.3-2.3482

8. Lano K., Xue Q., Haughton H. A Concrete Syntax Transformation Approach for Software Language Processing. SN Computer Science. 2024. Vol. 5. Article 645. DOI: 10.1007/s42979-024-02979-y

9. Ražinskas M., Miliūnas B., Jurgelaitis M., Čeponienė L., Bisikirskienė L. Transforming Sketches of UML Use Case Diagrams to Models. IEEE Access. 2024. Vol. 12. C. 185826–185837. DOI: 10.1109/ACCESS.2024.3514455

10. Maschotta R., Silatsa N., Jungebloud T., Hammer M., Zimmermann A. An OCL Implementation for Model-Driven Engineering of C++. Lee R. (ed.) Software Engineering Research, Management and Applications (SERA 2022). Studies in Computational Intelligence. Vol. 1053. Springer, Cham, 2022. C. 151–168. DOI: 10.1007/978-3-031-09145-2_10

**References**

1. Allian, A. P., Nakagawa, E. Y., Martinez, J., & Oliveira Jr., E., et al. (2022). Variability implementation and UML-based software product lines. In *UML-Based Software Product Line Engineering with SMarty*, pp. 27–40. Springer, Cham. https://doi.org/10.1007/978-3-031-18556-4_2

2. Cu, C., Ye, X., & Zheng, Y. (2019). Xlinemapper: a product line feature-architecture-implementation mapping toolset. In *41st International Conference on Software Engineering: Companion Proceedings (ICSE '19)*, pp. 87–90. IEEE Press, Piscataway. https://doi.org/10.1109/ICSE-Companion.2019.0004516

3. Abdelmadjid, L. (2022). Uncertain decision-making requirements formalizing with CF UML model. *Procedia Computer Science*, 192, pp. 2503–2512. https://doi.org/10.1016/j.procs.2021.12.247

4. Cazzola, W., & Olivares-Corichi, I. (2022). Bridging the model-to-code abstraction gap with fuzzy logic: FLiRTS 2 for UML class diagrams. *Software and Systems Modeling*, 21, pp. 1717–1742. https://doi.org/10.1007/s10270-021-00899-6

5. Cardiel-Ortega, J. J., López-Robles, J. R., Otegi-Olaso, J. R., & Gamboa-Rosales, H. (2024). Probabilistic fuzzy system for evaluation and failure modes in FMEA. *Processes*, 12(6), Article 1197. https://doi.org/10.3390/pr12061197

6. Dalpiaz, F., Ferrari, A., Franch, X., & Palomares, C. (2018). Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, 35(5), pp. 115–119. https://doi.org/10.1109/MS.2018.3571242

7. Yang, M., & Ban, A. (2024). Automated UML class diagram generation from textual requirements using NLP techniques. *JOIV International Journal on Informatics Visualization*, 8(3–2), pp. 1905–1915. https://doi.org/10.62527/joiv.8.3-2.3482

8. Lano, K., Xue, Q., & Haughton, H. (2024). A concrete syntax transformation approach for software language processing. *SN Computer Science*, 5, Article 645. https://doi.org/10.1007/s42979-024-02979-y

9. Ražinskas, M., Miliūnas, B., Jurgelaitis, M., Čeponienė, L., & Bisikirskienė, L. (2024). Transforming sketches of UML use case diagrams to models. *IEEE Access*, 12, pp. 185826–185837. https://doi.org/10.1109/ACCESS.2024.3514455

10. Maschotta, R., Silatsa, N., Jungebloud, T., Hammer, M., & Zimmermann, A. (2022). An OCL implementation for model-driven engineering of C++. In R. Lee (Ed.), *Software Engineering Research, Management and Applications (SERA 2022)*. *Studies in Computational Intelligence*, Vol. 1053, pp. 151–168. Springer, Cham. https://doi.org/10.1007/978-3-031-09145-2_10

*Дата першого надходження рукопису до видання: 22.09.2025*
*Дата прийнятого до друку рукопису після рецензування: 17.10.2025*
*Дата публікації: 28.11.2025*