УДК 004.42
DOI https://doi.org/10.32782/2663-5941/2025.1.2/23

Нікітченко М.І.
Національний університет «Одеська політехніка»

ДВОРІВНЕВА АРХІТЕКТУРА UML НА ОСНОВІ ГІБРИДНОГО ФОРМАТУ JSON І XMI

Стаття розкриває гібридний підхід до зберігання UML-моделей, що поєднує офіційний стандарт XMI з легким JSON-представленням. Такий формат дає змогу ефективно працювати з «базовими» елементами (класи, атрибути, операції) у вигляді JSON, а складні поведінкові аспекти (діаграми станів, діяльності тощо) відображати у фрагментах XMI. Запропонована дворівнева схема доповнена модулем валідації, покликаним забезпечити узгодженість між обома рівнями моделі та попередити розсинхронізацію.

У роботі наведено приклади інтеграції з популярними UML-редакторами (StarUML, EnterpriseArchitect, PlantUML) та IDE. Показано, як StarUML може інтерпретувати JSON із вбудованими XMI-вставками, а EnterpriseArchitect – працювати з повноцінним XMI, куди за потреби перетворюються JSON-дані «на льоту». Згадано типову послідовність використання: спочатку для швидких змін та прототипування задіяно «легкий» JSON, а з наростанням складності в модель додаються XMI-фрагменти з детальною поведінкою.

Автор детально обговорює переваги такого підходу. Серед них – прискорене прототипування, збереження формальної точності для складних частин, можливість безболісної генерації коду й здійснення реверс-інжинірингу. При цьому визнається, що дворівнева архітектура є складнішою в реалізації, потребує налаштування конвертерів для зв'язку JSON та XMI й може ускладнювати роботу над моделлю у великих командах, коли зміни паралельно вносяться в обидва формати. Також окремі інструменти не підтримують «частковий» імпорт XMI, що спонукає до створення додаткових модулів чи адаптерів.

У підсумку гібридний формат постає як компромісне рішення: він дає змогу зберігати потрібну деталізацію UML-моделей, не відмовляючись від простоти JSON-редагування. Це особливо актуально в сучасних проектах, що швидко розвиваються й потребують синхронізації між UML-діаграмами, IDE та фактичним кодом. Застосування такого підходу сприяє підвищенню ефективності розробки складних програмних систем, поєднуючи швидкість внесення змін і глибоку специфікацію поведінкових аспектів.

Ключові слова: *гібридний формат, JSON, XMI, інтеграція з IDE, автоматична генерація коду, валідація, розширення UML-інструментів, моделювання програмного забезпечення.*

Постановка завдання. Сучасні програмні системи часто вирізняються підвищеною складністю [1] та розподіленою структурою, що вимагає найефективніших підходів до їхнього проектування, аналізу та супроводу. У зв'язку з цим мова UML (Unified Modeling Language) стає найважливішим інструментом формалізації вимог і візуалізації архітектури [2-3]. У разі збільшення масштабу проектів постає питання оптимального зберігання і редагування UML-моделей, а також забезпечення їхньої інтеграції з різними інструментами розробки, що підкреслюється і в контексті Model-Driven Architecture [4-5]. На практиці істотними факторами виявляються як багатство формального опису, що дає змогу глибоко опрацьовувати аспекти поведінки та взаємодії компонентів, так і зручність швидкого внесення змін. Формати XMI і JSON, згадані раніше у роботі [6], по-різному задовольняють ці критерії: перший забезпечує деталізований опис UML-елементів, а другий спрощує ручне редагування. Однак жоден із них не дає універсального рішення, задовольняючи одночасно потреби у високій формальній точності та гнучкості інтеграції, особливо в умовах сучасних IDE. Звідси виникає завдання пошуку формату, здатного поєднувати формальну потужність XMI з легкістю і наочністю JSON. Актуальність цього завдання визначається безпосередньо потребою в прискоренні темпів розробки та скороченні ризиків, пов'язаних із неповнотою або неузгодженістю проектної документації.

Основна мета цієї роботи полягає в розробці архітектури, що дає змогу одночасно враховувати переваги XMI і JSON під час опису UML-моделей. Передбачається, що створюване рішення спростить процес редагування і, водночас, збереже

високий ступінь відповідності специфікації UML для складних сценаріїв, пов'язаних із поведінкою системи. З урахуванням сформульованої мети виникає необхідність розв'язати низку питань: яким чином структурувати дані, щоб у базових випадках зберігати зручність JSON, а для складних UML-елементів запроваджувати механізми деталізованого опису XMI; як адаптувати цю структуру до вже наявних UML-інструментів і середовищ розробки; а також як забезпечити баланс між формальною виразністю і швидкістю внесення змін.

Аналіз останніх досліджень і публікацій. Найвідоміші інструменти для роботи з UML-моделями істотно різняться за форматом зберігання, наборами підтримуваних діаграм і рівнем інтеграції із зовнішніми системами.

StarUML застосовує JSON-подібний формат, що полегшує внесення правок, але ускладнює опис складних аспектів поведінки. EnterpriseArchitect, спираючись на XMI, добре масштабується для великих проєктів, проте може бути громіздким у невеликих динамічних командах. PlantUML використовує текстовий синтаксис, що дозволяє швидко генерувати діаграми, але не повністю покриває специфікацію UML. У підсумку кожне з рішень спеціалізується на своєму завданні і не дає універсального балансу між докладним моделюванням і зручністю редагування.

Нижче наведено порівняльну таблицю ключових характеристик трьох популярних рішень (табл. 1).

Тим не менш, на практиці формат XMI часто виявляється великоваговим, тоді як JSON і текстові описи втрачають формальну повноту, особливо під час моделювання поведінки. Додатково IDE вимагають надійної синхронізації UML-моделей із кодом, що ускладнюється за відсутності єдиного стандартизованого представлення всіх аспектів. Усе це вказує на потребу в гібридному підході, що поєднує деталізовану специфікацію і простоту правок, а також враховує щоденну роботу розробників у сучаснихIDE.

Постановка завдання. Інструменти, засновані на XMI, розв'язують задачу перенесення даних між складними системами моделювання, але створюють труднощі в умовах швидких змін. Підходи, що спираються на JSON, вдалі на стадії прототипування, проте обмежені в описі поведінкових діаграм. Потрібен єдиний формат, що дає змогу гнучко керувати моделлю і підтримувати інтеграцію з IDE, не втрачаючи сумісності з великими CASE-засобами.

Для усунення перерахованих обмежень запропоновано гібридний формат, який зберігає базові елементи UML (класи, атрибути, операції) у JSON, який можна прочитати людиною, а розширені аспекти (діаграми станів, діяльності) – у фрагментах, які відповідають структурі XMI. Передбачається, що така архітектура забезпечить зручність редагування і дасть змогу автоматично генерувати формальну специфікацію для взаємодії з великими UML-інструментами.

Ключовими показниками ефективності стануть зручність ручного редагування, сумісність з EnterpriseArchitect і StarUML, а також здатність відображати поведінкові діаграми без втрати даних. Обмеження полягають в ускладненій процедурі парсингу гібридного формату та потенційній необхідності адаптації в IDE, що не підтримують частковий імпорт XMI.

Виклад основного матеріалу. Як уже було викладено в [7], ідея полягає в тому, щоб зберігати просту інформацію в JSON, а складні елементи UML передавати у вигляді XMI-вставок. Нижче наведено приклад, де клас «Example» описується в JSON, а для асоціацій використовується XMI-фрагмент.

Таблиця 1

Порівняння інструментів для роботи з UML

Інструмент	Формат збереження	Основна область використання	Рівень інтеграції	Переваги / обмеження
StarUML	JSON (пропрієтарна структура)	Швидке прототипування та редагування	Базові плагіни для IDE	Простий для читання формат; можлива підтримка редагування скриптів. Обмежено деталізація моделей.
Enterprise Architect	XMI (у різних варіантах)	Великі корпоративні проекти, детальні UML-діаграми	Розширені плагіни та API	Глибока підтримка UML специфікації; широкі можливості реверс-інжинірингу. Важко редагувати вручну.
PlantUML	Текстовий формат (псевдо-UML)	Візуалізація та вбудовування у CI/CD, IDE	Широкі плагіни та додаткові розширення.	Зручний для створення діаграм за текстовими скриптами; слабо підходить для просунутої специфікації UML.

```
{
  "_type": "UMLClass",
  "name": "Example",
  "attributes": [
    {
      "name": "username",
      "type": "String"
    }
  ],
  "complexRelations": "<xmi:Relationxmi:type=
\"uml:Association\" source=\"class1\" target=
\"class2\" />"
}
```

Завдяки цьому JSON залишається зручним для швидкого редагування та візуалізації, а XMI зберігає складні поведінкові діаграми. Щоб уникнути невідповідностей, у системі передбачають два рівні зберігання (JSON і XMI) і модуль валідації. Він стежить, щоб під час змін в одній частині моделі оновлювалися відповідні посилання в іншій.

Для сумісності зі StarUML достатньо доповнювати JSON XMI-вставками. EnterpriseArchitect імпортує та експортує XMI, куди за потреби конвертуються JSON-дані. PlantUML, з його текстовими діаграмами, здебільшого використовує базові структури з JSON і може ігнорувати фрагменти, що не розпізнаються. Завдяки цьому не потрібно переходити цілком на XMI, якщо розробка йде зручно в JSON, але за необхідності доступна повна специфікація UML.

В IDE зазвичай реалізують пряму генерацію коду і зворотне проектування. Гібридний підхід розширює ці можливості: базові елементи (класи і методи) зберігаються в JSON, а детальні поведінкові сценарії – в XMI. Для синхронізації створюють API або конвертер, який відображає зміни IDE в «легкій» частині і, за необхідності, оновлює XMI-фрагменти. Це дає змогу уникнути пошкодження складних діаграм і знижує ризик розсинхронізації, оскільки JSON і XMI обслуговуються паралельно, але пов'язані спільною моделлю.

В основі пропонованої системи лежить дворівнева модель, у якій перший рівень відповідає за роботу з JSON-описами базових UML-елементів, а другий – за зберігання розширених поведінкових конструкцій у форматі XMI. Ці рівні взаємодіють через модуль валідації, що перевіряє узгодженість і взаємозв'язки між сегментами. Поряд із ним вводять модуль інтеграції із зовнішніми застосунками (UML-інструментами та IDE), який забезпечує експорт та імпорт гібридних даних у відповідному для кожного інструменту поданні.

Умовно на схемі (Рис. 1) можна виділити шість функціональних блоків:

1. сховище JSON-даних, що містить інформацію про класи, інтерфейси і прості відносини;

2. сховище XMI-фрагментів, що містить детально описані діаграми станів, діяльності та інші складні елементи;

3. модуль обробки JSON, що виконує операції парсингу та серіалізації базових структур UML-моделі;

4. модуль обробки XMI, що відповідає за розбір і генерацію розгалужених поведінкових діаграм;

5. модуль валідації та синхронізації, що координує взаємодію між JSON- і XMI-рівнями, стежачи за несуперечливістю посилань і відповідністю специфікації UML;

6. модуль інтеграції із зовнішніми інструментами, що за потреби дає змогу трансформувати гібридний формат у «чистий» XMI або в JSON-структури для подальшого використання в StarUML, EnterpriseArchitect, PlantUML і IDE.

На практиці ці блоки являють собою взаємопов'язані компоненти, що працюють поверх загального представлення UML-моделі. Перехід між JSON і XMI здійснюється прозоро для основних сценаріїв, так що користувач або IDE можуть не помічати внутрішньої складної структури. При цьому формально кожне оновлення в одному рівні відображається в другому через механізми синхронізації та валідації.

У процесі роботи системи модулі здійснюють низку послідовних кроків. Коли користувач вносить зміни в структуру діаграми (наприклад, додає новий клас або атрибут), система оновлює JSON-сегмент, оскільки базові елементи зберігаються саме в ньому. Якщо ці зміни зачіпають поведінку (додавання станів або переходів), механізм звертається до XMI-рівня: він або створює нову XMI-структуру для новоствореної поведінки, або коригує вже наявну. Аналогічна схема застосовується під час завантаження моделі із зовнішнього джерела. Якщо імпортується «важкий» XMI-файл, модуль обробки XMI зберігає всі поведінкові діаграми в тому самому форматі, а базові відомості про класи та зв'язки передає в JSON-сегмент, перетворюючи їх на спрощений вигляд.

Роль модуля валідації полягає в тому, щоб під час кожного циклу змін упевнитися, що об'єкти, згадані в XMI, узгоджуються з їхніми «заглушками» в JSON. Наприклад, якщо в поведінко-

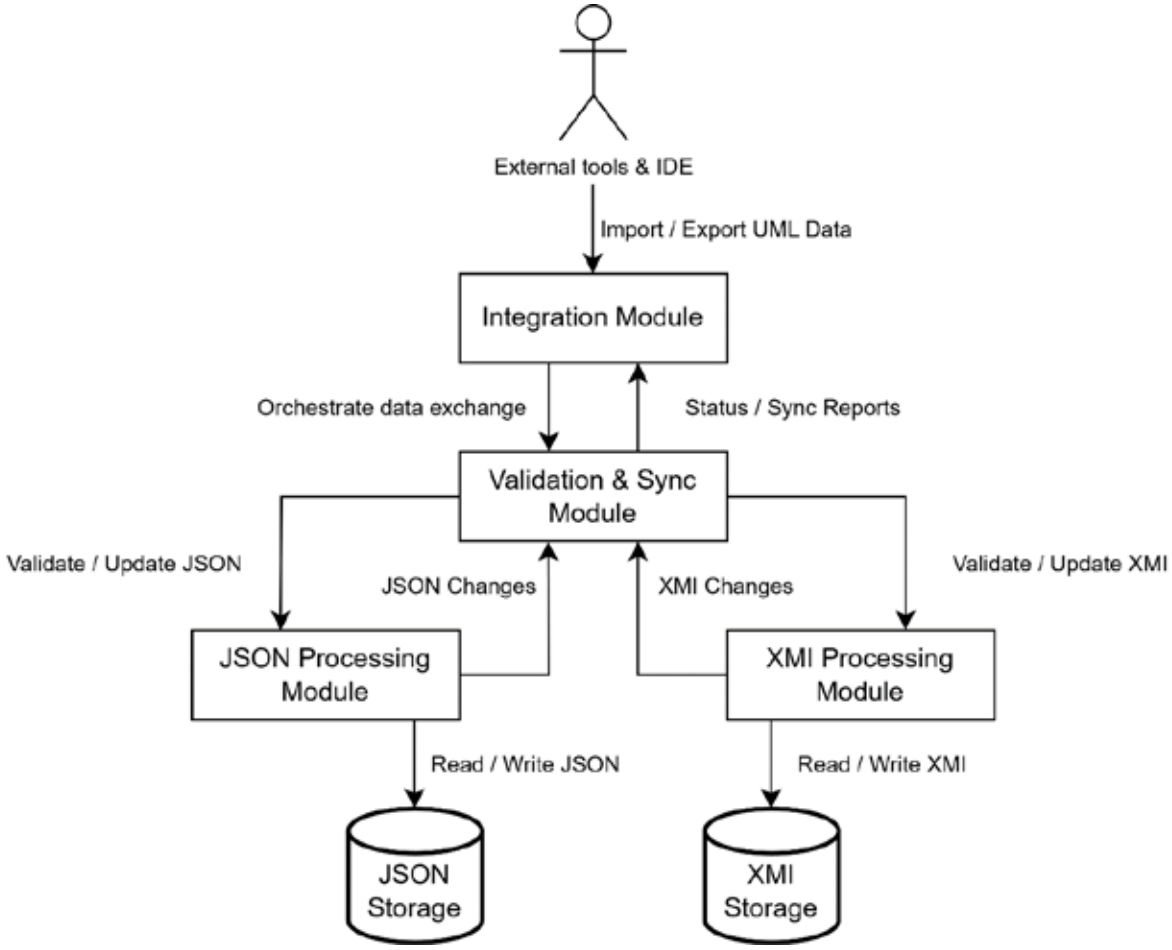

Рис. 1. Архітектурна модель системи

вій діаграмі XMI з'являється новий елемент, що відповідає методу класу, модуль перевіряє, чи є згадка про такий метод у JSON-частині; у разі потреби оновлює його або формує попередження про невідповідність.

На користувацькому рівні система може представлятися єдиним редактором UML-моделей. У графічному інтерфейсі або в текстовому (за використання тих самих PlantUML- або JSON-сумісних інструментів) користувач бачить елементарні структури в зручному для сприйнятті людиною форматі. Детальніші аспекти поведінки можуть відображатися або редагуватися в окремих вкладках або через спеціальні діалогові вікна, які фактично взаємодіють зі сховищем XMI-фрагментів.

Якщо йдеться про інтеграцію з IDE, користувач отримує можливість безпосередньо редагувати класи, методи та атрибути в коді, а система за рахунок внутрішніх конвертерів синхронізує зміни з гібридним форматом. За необхідності IDE може ініціювати звернення до поведінкових діаграм, використовуючи вже механізми XMI-обробки. Такий підхід дає змогу гнучко поєднувати переваги ручного редагування JSON-файлів і точну передачу семантики UML, що істотно полегшує процес проектування і скорочує ризик розбіжностей між моделлю і кодом.

В якості прикладу можна розглянути наступний сценарій використання. У типовій ситуації розробник починає роботу з моделлю в середовищі розробки, де вже встановлено необхідні плагіни для підтримки гібридного формату. Припустимо, що йому необхідно описати простий клас User з полем username і кількома методами. На етапі початкового прототипування модель представлена лише в JSON-файлі, оскільки потрібна тільки базова інформація про структуру класу. Приклад цього опису може мати такий вигляд:

```
{
  "_type": "UMLClass",
  "name": "User",
  "attributes": [
    {"name": "username", "type": "String"}
  ],
```

```
  "operations": [
    {"name": "validateUsername", "returnType":
"boolean"}
  ]
}
```

Розробник продовжує роботу і вирішує додати модель поведінки, що описує стани об'єкта класу User (наприклад, «Неактивний» і «Активний») і перехід «Активувати». У цей момент IDE або спеціалізований UML-редактор додає в проект XMI-фрагмент, що відповідає за реалізацію поведінкової діаграми. З урахуванням принципів гібридного зберігання з'являється додатковий блок, впроваджений у відповідну JSON-структуру, але відповідальний за складні аспекти моделі. Приклад (спрощений фрагмент) може виглядати так:

```
{
  "_type": "UMLClass",
  "name": "User",
  "attributes": [...],
  "operations": [...],
  "behaviorXMI": "<uml:StateMachinexmi:id=\
"sm¹\" name=\"UserStateMachine\">"
                 +
"<regionxmi:type=\"uml:Region\"
xmi:id=\"r1\">"
                 + "<subvertexxmi:type=\"uml:
State\" xmi:id=\"s1\" name=\"Inactive\"/>"
                 + "<subvertexxmi:type=\"uml:
State\" xmi:id=\"s2\" name=\"Active\"/>"
                 + "<transitionxmi:type=\"uml:
Transition\" xmi:id=\"t1\" name=\"Activate\" "
                 + "source=\"s1\"
target=\"s2\"/>"
                 + "</region>"
                 + "</uml:StateMachine>"
}
```

Далі розробник може візуалізувати цю діаграму в інструментах на кшталт StarUML, де JSON-частину відносно легко відобразити як звичну діаграму класів, а вбудований XMI-фрагмент буде інтерпретовано як кінцевий автомат у новій вкладці редактора. Якщо потрібно експортувати отриману модель в EnterpriseArchitect, загальний модуль інтеграції перетворює JSON-сегменти в XMI-опис на рівні класів, зберігаючи детальну інформацію про поведінкову діаграму безпосередньо з XMI-фрагмента. Завдяки такому механізму забезпечується повноцінна підтримка специфікації UML: «легка» частина залишається зручною для ручного редагування, а «важка» служить для формального опису та взаємодії з інструментами, орієнтованими на XMI.

На практиці це означає, що на ранній стадії проектування, коли потрібна тільки базова структура, розробники майже не торкаються XMI-фрагментів і оперують переважно JSON. У міру зростання вимог до функціоналу та ускладнення поведінки системи коду XMI стає більше, і UML-інструменти, які вміють аналізувати детальні діаграми, пропонують додаткові можливості (наприклад, перевірку коректності станів і переходів). Тим часом інтеграція з IDE не страждає: автоматична генерація коду ґрунтується на JSON-описі класів, а складніші елементи синхронізуються завдяки прозорим механізмам конвертації та валідації, вбудованим у гібридну архітектуру.

Таким чином, гібридний формат (JSON + XMI) дає можливість гнучко і формально описувати UML-моделі. На «легкому» рівні JSON забезпечує швидке виправлення та інтеграцію із зовнішніми інструментами, а для детальних поведінкових сценаріїв слугує XMI. Це підвищує продуктивність прототипування і спрощує впровадження в IDE, оскільки базові елементи легко синхронізуються з кодом, тоді як складні діаграми можуть рендеруватися в інструментах, що підтримують XMI.

Однак двоформатна структура ускладнює конфігурацію і вимагає додаткових механізмів валідації. Також не всі засоби допускають часткове читання XMI, що іноді потребуватиме додаткових конвертерів.

Подальше вдосконалення такого підходу передбачає тіснішу інтеграцію з IDE і розширене використання валідаційних механізмів. Бажано, щоб будь-які зміни у вихідному коді та XMI-складовій автоматично узгоджувалися з JSON-описом і навпаки. Додатково варто покращувати способи контролю версій та інструменти візуалізації, щоб користувач міг своєчасно бачити конфлікти та невідповідності.

Висновки. Запропонована гібридна архітектура поєднує зручність JSON із формальною точністю XMI, що особливо корисно під час проектування складних систем, які потребують швидкої ітерації та підтримки повноти UML-специфікації. Дворівнева схема зберігання, доповнена механізмами валідації, дає можливість розвивати систему без втрати деталей. При цьому залишається завдання адаптації гібридного формату до наявних інструментів і IDE.

Незважаючи на технічні складнощі, такий підхід відкриває шлях до більш тісної взаємодії між UML-моделлю і кодом, підтримуючи як прототипування, так і детальне моделювання, і тим самим підвищує ефективність розробки сучасних програмних систем.

Список літератури:

1. Tang A., Liang P., Vliet H. Software Architecture Documentation: The Road Ahead. *Ninth Working IEEE/IFIP Conference on Software Architecture*. 2011. P. 252–255. DOI: 10.1109/WICSA.2011.40.

2. Fowler M. UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language, 3rd Edition. Addison-Wesley Professional, 2018. 208 p.

3. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual, 2nd Edition. Boston: Addison-Wesley, 2005. Vol. 1. 721 p.

4. Brown, A. Model driven architecture: Principles and practice. *Software and System Modeling 3*. 2004. P. 314–327. DOI: 10.1007/s10270-004-0061-2.

5. Bezivin J., Gerbe O. Towards a precise definition of the OMG/MDA framework. *International Conference on Automated Software Engineering*: materials of 16th annual inter. conf. 2001. P. 273-280. DOI: 10.1109/ASE.2001.989813.

6. Нікітченко М. Управління та редагування UML-документів: структура, інтеграція, автоматизація. *Інформатика. Культура. Техніка*: матеріали X міжнар. наук.-прак. конф. Одеса, 2024. Т. 1. № 1. С. 104-111. DOI: 10.15276/ict.01.2024.15.

7. Нікітченко М. Розвиток і вдосконалення UML-моделей у гібридному форматі. *Традиційні та інноваційні підходи до наукових досліджень*: матеріали VIII міжнар. наук. конф. Дрогобич, 2025. С. 294-296. DOI: 10.62731/mcnd-31.01.2025 (у друці).

Nikitchenko M.I. TWO-TIER UML ARCHITECTURE BASED ON HYBRID JSON AND XMI FORMAT

The article reveals a hybrid approach to storing UML models that combines the official XMI standard with a lightweight JSON representation. This format makes it possible to work efficiently with "basic" elements (classes, attributes, operations) in the form of JSON, and display complex behavioral aspects (state diagrams, activities, etc.) in XMI fragments. The proposed two-level scheme is complemented by a validation module designed to ensure consistency between both levels of the model and prevent out-of-synchronization.

The paper provides examples of integration with popular UML editors (StarUML, Enterprise Architect, PlantUML) and IDEs. It is shown how StarUML can interpret JSON with built-in XMI inserts, and Enterprise Architect can work with full-fledged XMI, where JSON data is converted on the fly if necessary. A typical usage sequence is mentioned: at first, "light" JSON is used for quick changes and prototyping, and as complexity increases, XMI fragments with detailed behavior are added to the model.

The author discusses the advantages of this approach in detail. These include accelerated prototyping, preservation of formal accuracy for complex parts, the possibility of painless code generation, and reverse engineering. At the same time, it is recognized that a two-tier architecture is more difficult to implement, requires setting up converters for JSON and XMI communication, and can make it difficult to work on a model in large teams when changes are made to both formats in parallel. In addition, some tools do not support "partial" XMI import, which requires the creation of additional modules or adapters.

As a result, the hybrid format appears as a compromise solution: it allows you to maintain the necessary detail of UML models without giving up the simplicity of JSON editing. This is especially true in modern, fast-paced projects that require synchronization between UML diagrams, IDEs, and actual code. Using this approach helps to increase the efficiency of developing complex software systems by combining the speed of making changes and deep specification of behavioral aspects.

Key words: *hybrid format, JSON, XMI, integration with IDEs, automatic code generation, validation, extension of UML tools, software modeling.*