УДК 004.43

О. В. Годич[2], В. В. Пасічник[1], Ю. О. Прокопів[1], Н. Б. Чайківський[2]
[1]Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж,
[2]ТзОВ "Філден Менеджмент Сервісез Європа"

ФРАКТАЛЬНІ ОБ'ЄКТИ ЯК ЗАСІБ УНІФОРМНОГО ПРОЕКТУВАННЯ ФУНКЦІЙ ІНФОРМАЦІЙНИХ СИСТЕМ

© Годич О. В., Пасічник В. В., Прокопів Ю. О., Чайківський Н. Б., 2015

Побудова надійних програмних систем залишається однією з найскладніших проблем в області розроблення програмного забезпечення. Із нашого досвіду це пов'язано із неузгодженістю між архітектурою програмного запезпечення на мікро- (об'єкти) та макро- (системні компоненти) рівнях, а з погляду використання – із неузгодженістю між предметною областю та її сприйняттям користувачами різних рівнів. Ми вважаємо, що ці два фактори пов'язані між собою і можуть бути вирішені на рівні архітектури інформаційної системи. Фрактальні об'єкти є підходом до архітектури програмного забезпечення із орієнтацією на предметну область з цілісним об'єктно-орієнтованим архітектурним стилем в своїй основі. Формалізується спосіб для проектування й забезпечуються технічні засоби для побудови ресурсно-орієнтованих програмних систем із прозорою предметною (онтологічною) областю. Прозорість предметної області надає засоби різним зацікавленим сторонам (зокрема користувачам та розробникам) для використання і підтримки еволюції програмної системи.

Ключові слова: предметні онтології, архітектури інформаційних систем, проектування інформаційних систем, розроблення інформаційних систем.

Constructing reliable enterprise software systems remains one of the most difficult problems in Software Engineering. Our experience is such that from the engineering perspective this is attributed to an impedance mismatch between software architecture at the micro (objects) and macro (system components) levels, and from the utilisation perspective -- to an impedance mismatch between the business domain model and its perception by different system's stakeholders. We argue that these two factors are related, and can be resolved at the level of information systems architecture. Fractal Objects is an approach to software architecture with the domain orientation and holistic object-oriented architectural style at its core. It formalises a way for designing and provides technical means for constructing resource-oriented software systems with a transparent domain (ontological) model. The domain model transparency empowers all system stakeholders (including end-users and developers) to better utilise and support evolution of an enterprise software system.

Key words: domain ontologies, information system architectures, information systems design, information systems development.

Вступ. Загальна постановка проблеми

Відомо [1], що значна кількість інформаційних систем зазнають невдач ще на стадії розроблення або ж унаслідок суттєвого ускладнення внесення змін після їхнього впровадження. Розуміючи, що ця проблема має багато різних за природою та значущістю причин (зокрема, організаційні, неефективне управління вимогами тощо), в роботі виокремлюється аспект *конструювання (construction) програмного забезпечення.* Саме цьому аспекту приділяється найменше уваги в сучасних дослідженнях в галузі програмної інженерії і практичних реалізаціях складних програмно-алгоритмічних комплексів. Водночас він має визначальний вплив як на початкових етапах розроблення інформаційної системи, так і на етапах підтримки таких систем,

впроваджених у виробництво. Конструювання якісного програмного забезпечення повинно явно використовувати онтологію цільової предметної області. Зокрема це стосується етапів проектування на рівні конструкцій мови програмування та тестування, а також в користувацькому інтерфейсі інформаційної системи. Для досягнення цієї мети слід володіти відповідним теоретичним формальним апаратом й технологічним інструментарієм для втілення такого підходу на практиці.

Роберт Мартін, стверджуючи про важливість *конструювання* програмного забезпечення у своїй доповіді на конференції NDC-2012 [2], наголошує: "досі не було досягнуто жодних істотних покращень у підходах і методологіях конструювання програмного забезпечення порівняно з результатами об'єктно-орієнтованого аналізу в 1990-х роках. Сучасні інструменти і підходи конструювання програмного забезпечення ведуть до приховування реальних цілей інформаційної системи на рівні коду". Інакше кажучи, з програмного коду, поданого тією чи іншою мовою програмування, та структури проекту практично не можливо виокремити мету та функціональність інформаційної системи. З одного боку, численні програмні бібліотеки (наприклад, web чи object-relational mapping бібліотеки тощо) є ключовими в процесах розроблення програмного забезпечення, спрощуючи низькорівневі технічні аспекти, водночас повністю приховують реальну модель і цілі розробників інформаційних систем. Зазначена важлива проблема є інтуїтивно зрозумілою та має багато спроб вирішення.

У роботі Айвера Якобсона [8] подано підхід (OOSE, *object-oriented software engineering*) для проектування програмного забезпечення і реалізації базової бізнес-функціональності як набору варіантів використання, що є незалежними від конкретних технологій, таких як бібліотека побудови веб-застосувань чи протокол комунікації. Ціллю такого підходу є забезпечення зручного переходу базової функціональності до оновленої або іншої технології побудови програмного забезпечення. Однак підхід Якобсона має два суттєві обмеження:

1. Частина основної функціональності, що забезпечує взаємодію із програмними бібліотеками, істотно ускладнює процес розроблення і підтримки інформаційної системи.

2. Підхід, за яким реалізується варіантність використання предметної області, обмежує програмну систему тим, що вимагає дотримання жорстких поведінкових правил.

Перше обмеження змушує розробників змінювати функціональність взаємодії із програмними бібліотеками відразу при зміні таких бібліотек або ж внаслідок зміни основної бізнес-функціональності. Друге обмеження зумовлює приховування онтології предметної області від кінцевих користувачів, обмежуючи тим самим взаємодію між системою й кінцевими користувачами в межах наперед визначених варіантів використання. Це відповідає *розмовній метафорі*, як визначено в [7], яка є природною для процесо-орієнтованих програмних систем. Поширеність підходів, які орієнтовані на процеси, пов'язана з поширенням імперативної парадигми програмування і значно впливає на загальний дизайн програмних систем, зокрема, інтерфейсів користувача (user interface, UI).

Приховування бізнес-моделі ускладнює для кінцевих користувачів міркування про програмну систему й ускладнює комунікацію між різними зацікавленими сторонами. *Метафора моделі світу* [7] акцентує на важливості відображення моделі предметної області в користувацькому інтерфейсі інформаційної системи. Кінцеві користувачі при цьому можуть безпосередньо взаємодіяти з моделлю без проміжкових репрезентацій, притаманних розмовній метафорі. Річард Поусон дослідив застосування метафори моделі світу для розроблення інформаційних систем у працях [11, 12]. Результатом проведеного дослідження є створення архітектурного шаблону й однойменної бібліотеки за назвою Naked Objects. Архітектурний шаблон Naked Objects підсилює відомий патерн MVC, інкорпоруючи предметно-орієнтовану парадигму формування моделі (M), і підтримує автоматичне генерування UI (VC) із цієї моделі. Naked Objects належить до категорії предметно-орієнтованого проектування (domain-driven design) [6] завдяки використанню предметно-орієнтованого підходу до моделювання. Предметно-орієнтоване проектування є альтернативою підходу до системного проектування, який розробив Айвер Джейкобсон.

Концептуально довільна інформаційна система має три функції [10], а саме:

1. Пам’яті: збереження стану предметної області.
2. Інформування: надання інформації про стан предметної області.
3. Активності: виконання дій, що змінюють стан предметної області.

Для систем корпоративного програмного забезпечення зазначені функції є визначальними. Водночас ані OOSE, ані Naked Objects та й інші відомі предметно-орієнтовані підходи не дозволяють якісно вирішувати проблему конструювання програмного забезпечення, що забезпечувало б можливість уніформного проектування та системної реалізації трьох вищезазначених функцій.

У статті подано оригінальний уніформний підхід, названий авторами Fractal Objects, що забезпечує предметно-орієнтований спосіб конструювання інформаційних систем із системним врахуванням функцій пам'яті, інформування й активності. Висвітлено основні принципи запропонованого авторами підходу з погляду *парадигми інженерії підприємства* [3].

Fractal Objects: приципи та архітектура

Концепція *спільного моделювання* у контексті моделювання інформаційних систем передбачає залучення всіх зацікавлених учасників до процесів моделювання. Загалом концепція має підтримуватись саме тією технологією, що використовується для конструювання конкретної інформаційної системи. В такому випадку головною метою фрактальних об'єктів на концептуальному рівні є надання *семантичної прозорості* моделі предметної області на різних рівнях абстракції: від програмного коду до інтерфейсу користувача. Метою досягнення семантичної прозорості є покращення взаємодії між різними зацікавленими в середовищі відповідної інформаційної системи особами. Отже, зафіксувавши об'єкт предметної області (Person, Wagon, Vehicle тощо), поданий кінцевому користувачеві як частина користувацького інтерфейсу, розробник, вочевидь, розуміє, про який саме об’єкт на рівні програмного коду йдеться. Це дає змогу користувачам, аналітикам і розробникам системи краще розуміти один одного як під час розроблення нової функціональності, так і при виправленні помилок на різних етапах життєвого циклу інформаційної системи. Важливо при цьому не обмежувати семантичної прозорості виключно об’єктами предметної області, а забезпечити її і для функціональних аспектів предметної області, репрезентованих процесами і діями, які дозволено використовувати у системі.

Під семантичною прозорістю розуміємо ізоморфізм між різними поданнями об’єктів предметної області (користувацький інтерфейс, код тощо), який природно генерується завдяки використанню запропонованого авторами підходу до конструювання інформаційних систем. Ми стверджуємо, що цього досягають, використовуючи *інваріантну репрезентацію* для моделювання і реалізації функцій пам’яті, інформування й активності.

Обчислювальне середовище

Відомо, що концептуальна модель даних визначається двома мовами: мовою опису даних і мовою маніпулювання даними [9]. Це визначення є коректним зокрема і для реляційного підходу. Обидві мови реалізовано у вигляді обширного інструментарію Structured Query Language (SQL), що підтримує як опис, так і маніпулювання даними. Забезпечуючи дотримання моделі даних, що описується в термінах SQL, можна зручно мігрувати між різними СКБД. Водночас слід зазначити, що реалізації конкретних СКБД є дуже специфічними, – вони реалізують оригінальні алгоритми і методи оптимізації для ефективного управління даними, які описано загальними моделями. Цей факт підтверджує властивість універсальності концептуальних моделей даних, що є бажаною для будь-якої інформаційної системи.

Автори припускають, що схожу концептуальну модель можна ефективно застосовувати для моделювання інформаційних систем з використанням об'єктно-орієнтованих мов програмування. В такій моделі присутня мова опису предметної області (як структурних, так і функціональних аспектів) та мова маніпулювання артефактами, які було отримано в результаті такого опису. Водночас передбачається наявність обчислювального середовища, в якому реалізується довільна модель предметної області.

Такий підхід можна порівняти з генеративним підходом до програмування [16]. Слід зазначити, що присутня чітка відмінність: генеративне програмування базується на генеруванні коду, який у багатьох випадках вимагає надалі змін. При цьому зміни, які вносяться у згенерований код для досягнення необхідної функціональності, спричиняють невідповідність вихідної моделі, яку було використано для генерування коду та остаточного варіанта інформаційної системи. Тому аналогія з SQL більш релевантна – зазначимо, що код SQL інтерпретується, при цьому той код, що згенерований конкретною СКБД, не зазнає жодних змін з боку розробника системи.

Метою функціонування обчислювального середовища є абстрагування від низькорівневих аспектів технологій, необхідних для виконання кінцевого програмного коду. Автори сформулювали вимоги щодо обчислювального середовища, дотримання яких забезпечило б надійність перевірки відповідної гіпотези. Найважливішими з них є:

- залежність між “середовищем” й “моделлю” підтримується в сенсі “розуміння” середовищем мов опису й маніпулювання, які використовуються для моделювання предметної області;
- “середовище” наділене інструментами відокремлення деталей збереження та механізмів комунікації, що забезпечує можливість виконання функцій інформаційної системи на одному комп’ютері, в локальній мережі або в мережі Інтернет;
- “середовище” повинно забезпечувати автоматизоване генерування інтерфейсу користувача на основі відповідної моделі для дотримання принципу прямої маніпуляції та досягнення ефективної людино-машинної взаємодії [13, 14].

Перша із наведених вимог вказує на необхідність *слабкого зв'язку* між моделями предметних областей й обчислювального середовища, яке може їх виконувати. Під слабким зв’язком розуміємо те, що зміни у внутрішніх аспектах реалізації обчислювального середовища не повинні вимагати змін моделі предметної області. Метою другої вимоги є абстрагування низькорівневих технічних деталей від розробників програмного забезпечення, аби вони могли зосередитися безпосередньо на моделюванні предметної області. Остання вимога акцентує на тому, що обчислювальне середовище повинно вирішувати проблему взаємодії користувачів зі системою так, щоб забезпечити семантичну прозорість з погляду використання системи користувачем.

Отже, двома основними складовими технології фрактальних об’єктів є спосіб опису моделі предметної області й обчислювальне середовище для виконання цих моделей. Прототип підходу Fractal Objects було розроблено мовою програмування Java, хоча можна використати будь-яку іншу об'єктно-орієнтовану мову програмування загального призначення. Вибір мови ґрунтувався на тому, що Java досі є однією з найпоширеніших на практиці мов програмування для реалізації транзакційних бізнес-систем. Додатковим аспектом на користь використання Java є широкі можливості Reflection API, які уможливлюють інтроспекцію моделей предметної області під час виконання програми з метою здійснення їхнього динамічного семантичного аналізу.

Об’єкти-сутності та об’єкти-компаньйони

Фрактальний об'єкт є парою *об’єкта-сутності* й *об'єкта-компаньйона*. Об’єкт-сутність є структурою деякого об’єкта предметної області, що моделюється. На рівні коду об’єкт-сутність реалізовують за допомогою класу, який має атрибути стану (властивості) і метастану, але не містить реалізації поведінки об'єкта. Роллю об'єкта-компаньйона є забезпечення операцій, які можна виконувати з об’єктом-сутністю. На рівні коду об’єкт-компаньйон описується інтерфейсом, який має методи, але не має стану. Разом об’єкт-сутність і об’єкт-компаньйон утворюють елементарний будівельний блок для забезпечення інваріантного подання будь-яких структурних і функціональних аспектів цільової предметної області.

Атрибути стану, що інкапсулюються в об’єкті-сутності, можуть бути простого типу (дата, число тощо) або мати тип об'єкта-сутності. Це дає змогу моделювати чітко визначені відношення між об'єктами-сутностями (один-до-одного, багато-до-одного або один-до-багатьох). Важливо зауважити, що з погляду класової ієрархії ієрархія об’єктів-сутностей є плоскою – складності поведінки і структурі досягають за допомогою композиції, а не успадкування. Зміни асоціацій між

об'єктами-сутностями під час виконання досягають простою зміною значення відповідних властивостей – це такий аспект, який не може бути реалізований через механізм успадкування.

Об'єкти-компаньйони забезпечують чотири універсальні операції, які можна виконати відносно об'єкта-сутності: *створення*, *зберігання*, *видалення* і *запит*. Ці операції концептуально описують, що можна зробити з екземпляром сутності. Об'єкт-сутність може бути створений, збережений, видалений або включений в пошуковий запит. Ці операції використовуються як примітиви для реалізації бізнес-логіки інформаційної системи.

В архітектурному стилі REST [5] для веб-застосунків було продемонстровано переваги підходу, в якому є обмежена кількість операцій. На наше переконання, таких переваг також досягають на рівні моделювання предметної області. Існує багато семантичних аналогій між стилем REST для веб-ресурсів (на макрорівні) й моделюванням об'єктів предметної області (на мікрорівні). Наприклад, використання URI шляхів для переходів між веб-ресурсами аналогічне використанню шляхів для переходів між об'єктами-сутностями в об'єктно-орієнтованому графі. Це узгоджується з оригінальним розумінням об'єктно-орієнтованих систем в інтерпретації Алана Кея, в якому звертається увага на схожість між об'єктами й мережами комп'ютерів [4]. І хоча веб є механізмом доставки повідомлень (а програмне забезпечення може працювати через веб або не через веб), його внутрішня природа з'єднаних між собою ресурсів із єдиним механізмом взаємодії добре підходить для об'єктно-орієнтованого моделювання. Це був один із ключових використаних нами принципів проектування для моделювання складних структур і поведінки в межах підходу із фрактальними об'єктами.

Подібно до об'єктів-сутностей об'єкти-компаньйони можна скомпонувати з метою досягнення складнішої поведінки. Вони забезпечують чітке відмежування поведінки від об'єктів-сутностей, коли об'єкти-сутності репрезентують структуру асоціацій між об'єктами в предметній області і несуть метаінформацію для опису моделі.

З огляду на те, що об'єкти-компаньйони є інтерфейсами, виникає природне запитання про те, які класи і на якому етапі виконуватимуть фактичну реалізацію. У нашій роботі обчислювальне середовище забезпечує загальну реалізацію об'єктів-компаньйонів, від якої можуть бути успадковані конкретні об'єкти-компаньйони й розширені в межах конкретної інформаційної системи. Усі посилання на інтерфейси, що описують об'єкти-компаньйони, прив'язуються до відповідних реалізацій за допомогою механізму *injection of control* (IoC) у конкретній інформаційній системі.

Як можна зрозуміти з визначення фрактального об'єкта, мовою опису моделі по суті є підмножина мови Java примітивів, які об'єднані за відповідною схемою і є *внутрішньою предметно-орієнтованою мовою* (*internal DSL*) [15]. Система типів Java разом із метаданими (на основі механізму анотацій) надає гнучкі засоби для опису і верифікації моделі предметної області у статично типізований спосіб. Java забезпечує можливість аналізу метаданих за допомогою процесорів анотацій, які інтегруються безпосередньо у процес компілювання. Отже, реалізований у такий спосіб семантичний аналіз моделі предметної області може повідомити про конкретні семантичні помилки на етапі компілювання. Метадані використовуються як декларативний механізм прив'язки фрактальних об'єктів з різними бізнес-правилами, такими як валідація, безпекові правила, управління розмежуванням транзакцій, візуальне подання об'єктів-сутностей тощо.

У фрагменті коду нижче подано приклад об'єкта-сутності, яка моделює особу в деякій предметній області (`Person`). Рядок 1 декларує, що особа може бути збережена в базі даних (`@Persistent`), рядок 2 вказує на відповідний об'єкт-компаньйон `IPerson`. Рядок 3 вказує, що `Person` розширює клас `AbstractEntity`, який надається фреймворком і забезпечує функціональність, необхідну для інтерпретації моделі. Параметр типу в `AbstractEntity` вказує тип унікального бізнес-ключа для об'єкта `Person`. Мінімальною вимогою для збереження стану кожної сутності є існування бізнес-ключа, який може бути простим або складеним (композитним). Рядки від 5 до 9 декларують примітивну властивість `birthday`, яка є збережуваною (@Persistent),

має певну валідаційну логіку (`@BeforeChange`), що реалізована у вигляді класу `BirthdayValidator` і застосовується щоразу при зміні властивості `birthday`. Рядок 8 декларує назву властивості та її опис (`@Title`), що автоматично використовується наскрізно в інформаційній системі, зокрема в інтерфейсі користувача, у повідомленнях про помилки, для логування тощо.

```
1 @Persistent
2 @CompanionObject(IPerson.class)
3 public class Person extends AbstractEntity<String> {
4     ......................
5     @IsProperty
6     @Persistent
7     @BeforeChange(@Handler(BirthdayValidator.class))
8     @Title(value = "Birthday", desc = "The date when person was born")
9     private Date birthday;
10    ......................
11 }
```

З метою реалізації трьох основних функцій інформаційної системи з використанням тієї самої інваріантної репрезентації, підхід Fractal Objects передбачає три види фрактальних об'єктів:

- збережувані сутності (реалізують функцію пам'яті) – фрактальні об'єкти, що моделюють сутності предметної області, які повинні підтримувати збережуваний стан;
- синтезовані сутності (реалізують функцію інформування) – фрактальні об'єкти, що синтезуються з декількох збережуваних сутностей або є підмножинами їхніх властивостей;
- функціональні сутності (реалізують функцію активності) – фрактальні об'єкти, що використовуються для моделювання дій і бізнес-процесів інформаційної системи; функціональна сутність містить одну або більше збережуваних, синтезованих або інших функціональних сутностей.

У переважній більшості проектів інформаційних систем вимагається, щоб більшість сутностей предметної області були збережуваними для подальшого використання й аналізу. Такі сутності моделюються за допомогою так званих *persistent* типів (збережуваних), які визначаються анотацією `Persistent`. Однак не всі сутності повинні бути збережувані. Однією з унікальних концепцій інформаційного моделювання за допомогою фрактальних об'єктів є те, що *синтезовані* і *функціональні* об'єкти надають можливість моделювати сутності точно в такий самий спосіб, як і *збережувані*. Ці види сутностей можуть змішуватись в різних комбінаціях, що є особливо зручним, коли існує необхідність об'єднати незалежні сутності в єдину бізнес-концепцію або варіант використання. Це забезпечує уніформний спосіб реалізації бізнес-процесів, у яких можуть брати участь всі три види фрактальних об'єктів.

Розглянемо приклад моделювання процесу заміни складових вагона у предметній області залізничних перевезень за допомогою функціональної сутності. Нехай об'єкти-сутності `Wagon`, `Bogie` та `Wheelset` є збережуваними об'єктами-сутностіми, які моделюють сутності предметної області вагон, візок і колісну пару відповідно. Вони можуть бути створені, збережені, видалені й витягнуті з бази даних. При цьому колісна пара (`Wheelset`) може бути приєднана (від'єднана) до візка (`Bogie`), а візок може бути приєднаний (від'єднаний) до вагона (`Wagon`). Кілька візків можуть бути приєднані та від'єднані до одного і того самого вагона під час одного сервісного обслуговування вагона (***wagon service)***. Екземпляри об'єкта-сутності `Bogie`, які були від'єднані, повинні бути деасоційовані з відповідним екземпляром `Wagon`, а ті, що були прикріплені – мають створити відповідні асоціації. Декілька таких замін повинні утворювати єдину транзакцію за участю модифікації декількох сутностей предметної області.

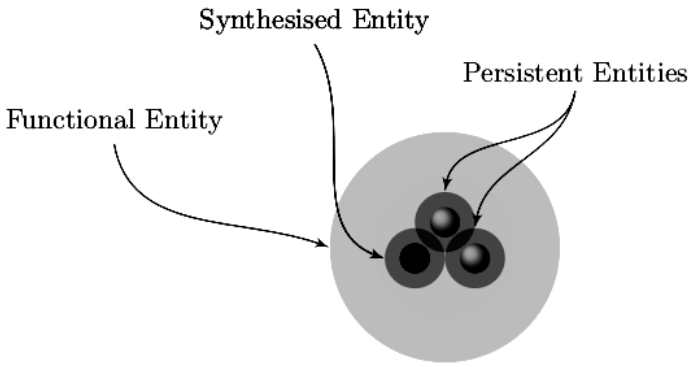
Звертаємо увагу, що досі було задекларовано три сутності. Але опис предметної області неявно містить додаткову сутність – сервісне обслуговування вагона (***wagon service***), яку можна змоделювати за допомогою функціонального виду фрактальних об'єктів. Такий функціональний

фрактальний об'єкт `WagonService` може мати набір модифікованих екземплярів `Bogies` і `Wagon` як його стан. Екземпляр `Wagon` і час проведення сервісу формуватимуть унікальний композитний бізнес-ключ. Фрагмент коду ілюструє цю функціональну сутність.

```
1 @CompanionObject(IWagonService.class)
2 public class WagonService extends AbstractEntity<Wagon> {
3
4     @IsProperty
5     private Set<Bogie> bogies;
6     ....................
7 }
```

Асоційований з об'єктом-сутністю `WagonService` об’єкт-компаньйон `IWagonService` у своїй реалізації повинен мати перевизначену операцію збереження, яка визначає демаркацію транзакції і виконує збереження змін до відповідних екземплярів типу Bogie і Wheelset за допомогою їхніх об'єктів-компаньйонів.

Зауважимо, що синтезовані і функціональні об’єкти-сутності можуть мати також збережувану природу. У прикладі зі сервісним обслуговуванням вагонів об'єкт-сутність `WagonService` може бути збережений з асоційованими `bogies` та `wagon`. З погляду побудови програмного забезпечення такі зміни до фрактальних об’єктів можуть бути застосовані ітеративно: будь-яка функціональна сутність може стати збережуваною одразу, як тільки така необхідність виникне.



Гетерогенна композиція фрактальних об’єктів

У своїй основі всі три види фрактальних об'єктів ідентичні, за винятком деякої інформації в метаданих, і можуть бути скомпоновані без будь-яких обмежень для формування складних структур та поведінки. При цьому обчислювальне середовище автоматично інтерпретує зміни в моделі. На рисунку схематично зображено функціональний об'єкт-сутність, який складається з одного синтезованого і двох збережуваних об'єктів-сутностей. Коли операція збереження застосовується до такого фрактального об'єкта-сутності, виконуються дії, які можуть змінити стан збережуваних сутностей, наприклад, так, як це було продемонстровано у прикладі зі сервісним обслуговуванням вагонів.

Висновки та перспективи подальших наукових розвідок

У статті висвітлено найважливіші принципи і технологічні інновації, які покладено в основу фрактальних об'єктів, а також їхнього застосування до реалізації функцій активності та пам’яті інформаційних систем. Реалізація функції інформування та предметно-орієнтовану побудову інтерфейсів користувача буде подано у наступних статтях. Серед найістотніших характеристик розробленого підходу слід виділити такі:

- уніформна модель розроблення та архітектури підносять концептуальний рівень програмування, надаючи можливість розробникам програмного забезпечення зосередитися на розробленні бізнес-рішень, замінивши програмування з використанням низькорівневих технологій на процес моделювання зі збереженням використання мови загального призначення;
- повний цикл конструювання інформаційних систем, починаючи від створення структур даних і завершуючи побудовою користувацького інтерфейсу, використовує єдину систему понять, яка є критично важливою для обчислювального середовища і спільного розуміння системи усіма її учасниками (від бізнесу до розробників).
- семантична прозорість й уніформне використання предметної області як на рівні розроблення, так і на рівні інтерфейсів користувача, сприяють порозумінню між розробниками програмного забезпечення, експертами предметної області та користувачами.

Сучасним етапом досліджень є розроблення предметно-орієнтованої системи типів з метою виявлення помилок у сенсі семантики предметної області і структурних помилок під час композиції фрактальних об'єктів. Також важливо проводити експерименти з метою оцінювання продуктивності конструювання інформаційних систем і ступеня задоволення кінцевих користувачів систем на основі технології фрактальних об'єктів.

1. Barker T. ERP implementation failure: a case study / T. Barker, M. Frolick // Information Systems Management. – 2003. – Vol. 20(4). – P. 43–49. 2. Martin R. Clean Architecture / R. Martin // Norwegian Developers Conference. – 2012. 3. Dietz J. The discipline of enterprise engineering / J. Dietz, J. Hoogervorst // Organisational Design and Engineering. – 2013 – Vol. 3(1). – P. 86–114. 4. Key A. The Early History of Smalltalk / A. Key // ACM SIGPLAN Notices. – 1993. – Vol. 28(3). – P. 69–95. 5. Fielding R. T. Architectural styles and the design of network-based software architectures / R. T. Fielding – University of California, Irvine, USA, 2000. – 180 p. 6. Haywood D. Domain-Driven Design Using Naked Objects / D. Haywood – Pragmatic Book-shelf, 2009. – 413 p. 7. Hutchins E. Direct Manipulation Interfaces / E. Hutchins, J. Holland, D. Norman // User Centered System Design: New Perspectives on Human-computer Interaction – 1985. – Vol. 1. – P. 311–338. 8. Jacobson I. Object Oriented Software Engineering: A Use Case Driven Approach / I. Jacobson - Addison-Wesley Professional, 1992. – 552 p. 9. Kalinichenko, L. A. Methods and tools for integrating heterogeneous databases, 1983, Nauka (in Russian), 1983. 10. Olive, A. Conceptual Modeling of Information Systems, 2007, Springer. 11. Pawson, R. and Matthews, R. Naked objects: a technique for designing more expressive systems, SIGPLAN Notices, V.36(12), pp.61–67, 2001. 12. Pawson, R., Naked Objects, Ph.D Thesis, 2004, Trinity College, Dublin, Ireland. 13. Shneiderman, B. The future of interactive systems and the emergence of direct manipulation. Behaviour and Information Technology, 1(3):237256, 1982. 14. Shneiderman, B. Direct manipulation: A step beyond programming languages. Computer, 16(8):5769, 1983. 15. DslBoundary, http://martinfowler.com/bliki/DslBoundary.html. 16.Чарнецки К. Порождающее программирование: методы, инструменты, применение. Для профессионалов [Текст] / К. Чарнецки, У. Айзенекер. – СПб.: Питер, 2005. – 736 с