переходу до нечітких чисел та зворотну від нечітких чисел до чітких. Роботу алгоритму проілюстровано двома прикладами наближення заданої послідовності з 10 імпульсів 20 та 100 спайк-послідовностями.

1. Maass W. Networks of spiking neurons: The third generation of neural network models / W. Maass // Neural Networks – 1997. – Vol. 10, No. 9. – P. 1659–1671. 2. Бодянский Е.В. Гетерогенная спайк-нейронная сеть с латеральными связями в задаче кластеризации / Е.В. Бодянский, А.И. Долотов // Системи обробки інформації. – 2007. – Вип. 8 (66). – С. 10–15. 3. Kasinski A. Comparison of Supervised Learning Methods for Spike Time Coding in Spiking Neural Networks / A. Kasinski, F. Ponulak // Int. J. Appl. Math. Comput. Sci. – 2006. – Vol. 16, No. 1. – P. 101–113. 4. Carnell A. Linear Algebra for Time Series of Spikes [Електронний ресурс] / A. Carnell, D. Richardson // University of Bath, UK. – 2002. – 7 p. – Режим доступу: esann.pdf. 5. Корн Г. Справочник по математике для научных работников и инженеров / Г. Корн, Т. Корн. – М.: Наука, 1970. – 720 с.

УДК 004.9

Н.Б. Шаховська
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

РОЗРОБЛЕННЯ АРХІТЕКТУРИ ПРОСТОРУ ДАНИХ ТА ПРОЕКТУВАННЯ КЛАСУ КЕРУВАННЯ ЙОГО КОМПОНЕНТАМИ

Побудовано архітектуру простору даних. Описано особливості налагодження функціонування простору даних як складної системи.

Ключові слова: простір даних, транслятор запитів, метамова.

The construction of architecture data space. Specifics debug operation data space as a complex system.

Keywords: space data translator requests metalanguage.

Вступ

Термін “big data” (великі дані) останнім часом спричинив великий резонанс, оскільки прогнозована кількість інформації у 2015 р. становитиме 10^{24}. Десятиліттями компанії приймали рішення, ґрунтуючись лише на транзакційних даних, що зберігаються в реляційних базах даних. Крім цих, очевидно, дуже цінних даних існує низка нетрадиційних, менш структурованих джерел: журнали веб-серверів (logs), соціальні мережі, пошта, давачі, фотографії і т.д., які можуть бути використані для отримання достатньо корисної інформації. Тому виникає необхідність у інформаційній технології опрацювання різноформатних даних [3]. Простір даних – це блоковий вектор, що містить множину інформаційних продуктів предметної області, поділену на три блоки: структуровані, напівструктуровані та неструктуровані дані. Над цим вектором визначено відношення між компонентами, операції над ІП та предикати. У статті спроектовано архітектуру простору даних як інформаційної технології для опрацювання великих даних.

1. Архітектура, принципи функціонування

Архітектура простору даних складається з декількох рівнів та об’єднує рівні даних, моделей керування та метаописів (рис. 1).

Структуру модулів простору даних подано на рис. 2.

Рівень даних становить інформаційний простір простору даних та об’єднує інформаційні ресурси. На рис. 2. позначений як хмара джерел даних.

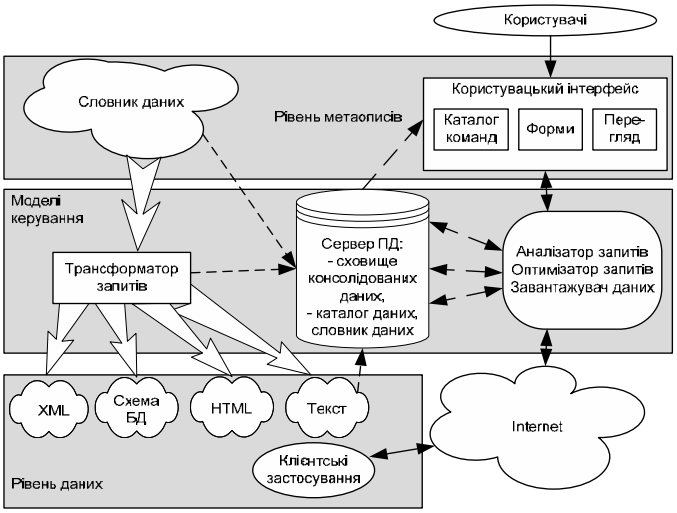

Рис. 1. Архітектура простору даних

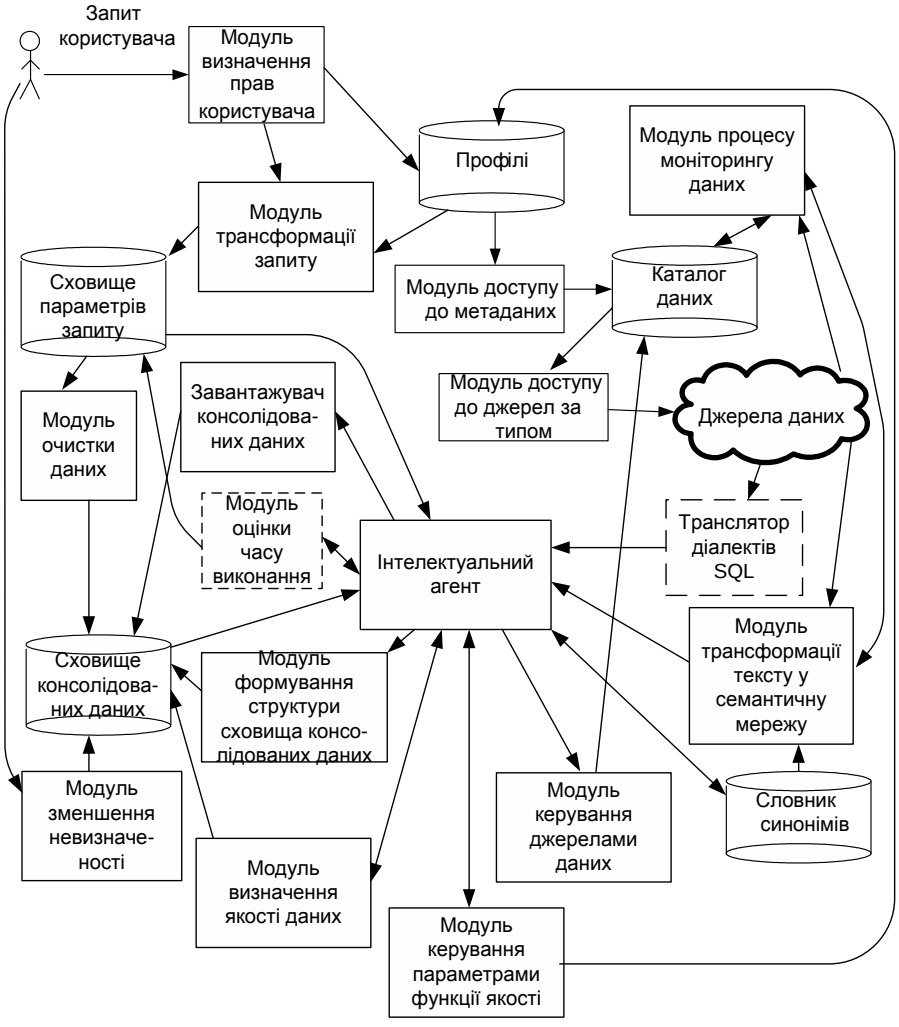

Рис. 2. Структура простору даних

Рівень моделей керування складається з модулів, що відповідають за організацію та управління простором даних. До них належать [1, 2]:

- модуль визначення прав користувача (процедура авторизації, побудована на основі операції визначення прав користувача);
- модуль трансформації запиту (на основі методу інтерпретації);
- модуль доступу до метаданих (на основі методу операції пошуку, побудована як операція запиту до метаданих;
- модуль доступу до джерела за типом – використання стандартних протоколів обміну даними;
- модуль трансформації тексту у семантичну мережу (на основі методу опрацювання напівструктурованих даних та алгоритмів аналізу текстових даних);
- інтелектуальний агент (на основі формального опису інтелектуального агента визначення структури джерела даних, алгоритму роботи інтелектуального агента);
- модуль формування структури сховища консолідованих даних – на основі методу побудови схеми сховища консолідованих даних та роботи інтелектуального агента визначення структури джерела даних;
- завантажувач консолідованих даних (на основі операції консолідації, методу консолідації даних),
- модуль очистки даних (на основі удосконалених операторів зрізу, оператора згортання, методу формування системи норм і критеріїв, методу аналізу, фільтрації та перетворення вхідних даних);
- модуль зменшення невизначеності (на основі методу застосування класифікаційних правил та модифікованого оператора усунення невизначеності у мережній структурі консолідованих даних);
- модуль визначення якості даних (на основі методу оцінювання якості консолідованих даних);
- модуль керування параметрами функції якості (на основі методики керування елементами простору даних на основі значення функції якості та рівнів довіри);
- модуль керування джерелами даних (на основі методики керування елементами простору даних на основі значення функції якості та рівнів довіри);
- модуль процесу моніторингу даних (на основі процесу моніторингу даних);
- модуль оцінювання часу виконання (на основі стандартних засобів фіксації часу виконання);
- транслятор діалектів SQL (на основі SQL-дескриптора).

Рівень моделей керування становить платформу підтримання ПД.

Рівень метаописів містить усю базову інформацію про джерела даних та методи доступу до них. Також визначаються методи опрацювання даних: для структурованих джерел – вибір, групування тощо, для напівструктурованих та неструктурованих – визначення структури або пошук за ключовими словами.

Окрім того, у просторі даних також передбачено сховища для зберігання профілів користувачів та тимчасового сховища параметрів запиту.

2. Технологічні аспекти

Для реалізації інтеграції даних з різнотипних джерел використано службу інтеграції SQL Server (SQL Server Integration Services, SSIS), яка надає гнучку і масштабовану архітектуру, що забезпечує ефективну інтеграцію даних у сучасному бізнес-середовищі.

SSIS складається з ядра підтримки потоку задач, орієнтованих на операції, і ядра підтримки потоку даних. Потік даних існує в контексті загального потоку задач (рис. 3).

В основу SSIS покладено конвеєр перетворення даних. Архітектура конвеєра підтримує буферизацію, що дозволяє конвеєру швидко працювати при маніпуляціях над наборами даних після

їх завантаження у пам'ять. Суть підходу полягає у виконанні усіх етапів ETL-процесу перетворення даних у межах однієї операції без проміжного зберігання даних, хоча специфічні вимоги до перетворення, операцій або засобів збереження можуть стати перешкодою при реалізації цього підходу.

Рис. 3. Схема інтеграції SSIS

SSIS за можливості навіть уникають копіювання даних у пам'яті. Це принципово відрізняється від традиційних ETL-засобів, які досить часто вимагають проміжного зберігання даних майже на кожному етапі процесу опрацювання та інтеграції даних. За допомогою SSIS усі типи даних (структуровані, неструктуровані, XML тощо) перед завантаженням у свої буфери перетворяться на табличну структуру.

Служба інтеграції SQL Server 2008 оптимізована для підключень через ADO.NET (попередні версії були оптимізовані для OLE DB або ODBC). Перехід на ADO.NET спрощує інтеграцію систем і підтримку третіми сторонами. Служби інтеграції SQL Server 2005 використовували OLE DB для виконання таких важливих завдань, як операції пошуку (lookups), але тепер для будь-яких завдань, пов'язаних з доступом до даних, можна застосовувати ADO NET.

У міру масштабування інтеграційного рішення часто продуктивність зростає лише до певної межі, а потім виходить на рівень, подолати який дуже важко. Служби інтеграції SQL Server 2008 знімають це обмеження за рахунок сумісного використання потоків (threads) множиною компонентів, що збільшує ступінь розпаралелювання і зменшує частоту блокування; це сприяє підвищенню продуктивності у великомасштабних системах із високим ступенем розпаралелювання на основі багатопроцесорних і багатоядерних апаратних платформ.

Пошук – одна з найпоширеніших операцій у інтеграційному рішенні. Служби інтеграції SQL Server 2008 прискорюють операції пошуку і дозволяють ефективно виконувати їх у великих таблицях. Завантажується повний кеш з будь-якого джерела, розмір кешу може перевищувати 4 Гб навіть у 32-розрядній операційній системі. Використовуючи частковий кеш, служби інтеграції SQL Server 2008 заздалегідь завантажують дані, необхідні для пошуку. Частковий кеш підтримує OLEDB, ADO.NET і ODBC для пошуку в базах даних, і відстежує попадання і промахи у процесі пошуку.

SSIS може витягувати (а також вивантажувати) дані з різних джерел, включаючи OLEDB, керовані джерела (ADO.NET), ODBC, плоскі файли, Excel, і XML за допомогою спеціального набору компонентів – так званих адаптерів (adapters). SSIS також може використовувати для отримання даних індивідуальні адаптери (custom adapters), тобто створені самостійно або іншими виробниками для своїх потреб. Це дозволяє включити успадковану логіку завантажувати дані безпосередньо до джерела даних, яке, своєю чергою, без додаткових дій може бути

впроваджене у потік даних SSIS. SSIS містить набір засобів перетворення даних, за допомогою яких можна робити з даними всі маніпуляції, які необхідні для побудови сховища консолідованих даних.

3. Структура класу керування компонентами ПД

Опишемо структуру класу керування компонентами ПД (рис. 4).

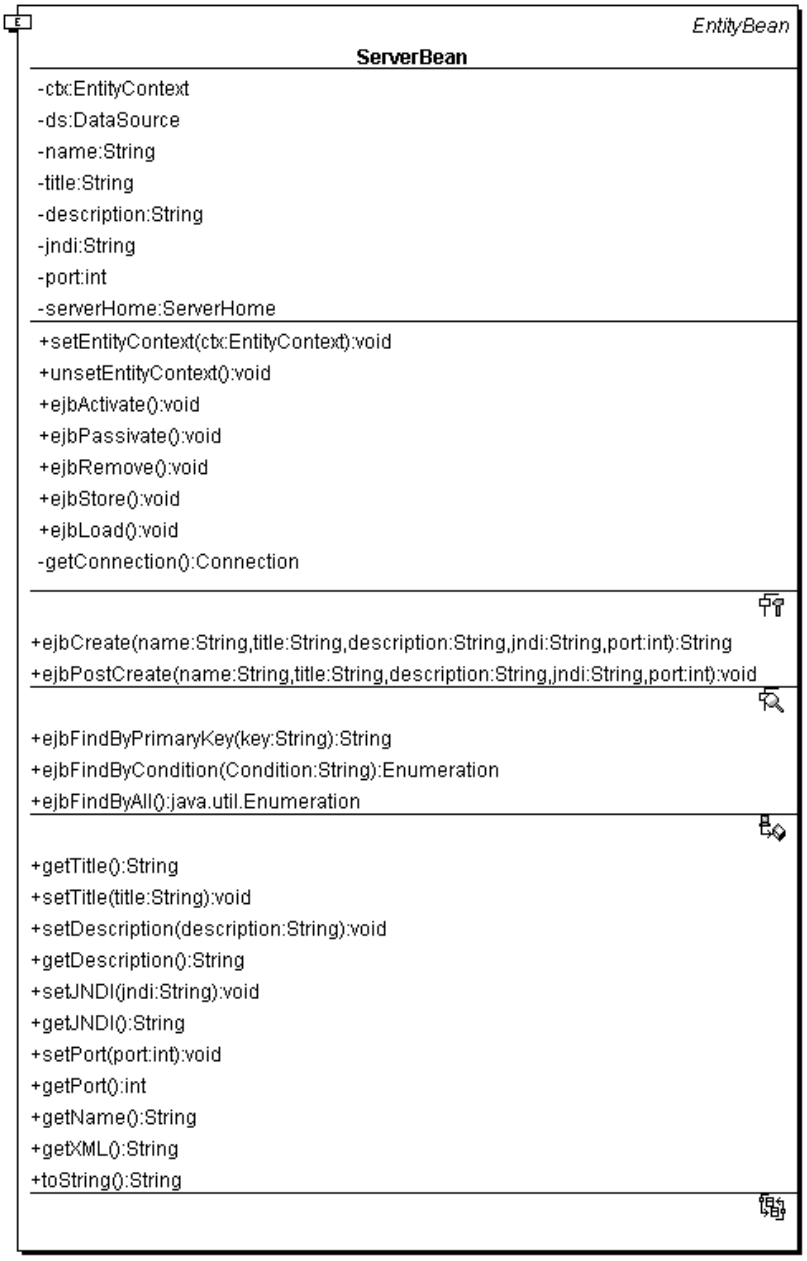

Рис. 4. Структура класу керування компонентами ПД

- ctx – посилання на об'єкт, що дає змогу компоненту отримувати службову інформацію про транзакції користувачів і дані про те, який користувач працює з компонентом;
- ds – посилання на пул з'єднань з базою даних;
- name, title, description, jndi, port – параметри компонента доступні через методи Remote-інтерфейсу;
- serverHome – посилання на Home-інтерфейс компонента Server;
- setEntityContext / unsetEntityContext – методи, в яких встановлюється ctx. Викликаються тільки контейнером;

• ejbActivate / ejbPassivate – методи, які керують життєвим циклом компонента. Викликаються тільки контейнером;

• ejbRemove – метод, який викликається перед знищенням компонента на стороні сервера (реалізує запит до бази даних на видалення цього компонента з бази);

• getConnection – метод, який викликають для з'єднання з пулом з'єднань. Його визначено більше для зручності, і до специфікації EJB він не має жодного відношення;

• ejbCreate – метод, який реалізує create-методи з Home-інтерфейсу. В ньому реалізують запити до бази даних для створення компонента і встановлюють параметри компонента;

• ejbPostCreate – методи викликаються після ejbCreate;

• ejbFind – методи реалізують методи пошуку, шукаються компоненти в базі даних;

• get/set – методи реалізують get / set методи, визначені в Remote-інтерфейсі;

• toString – визначений для більшої сумісності з інфраструктурою JAVA.

Метамодель діаграми інтелектуального агента зображено на рис. 5.

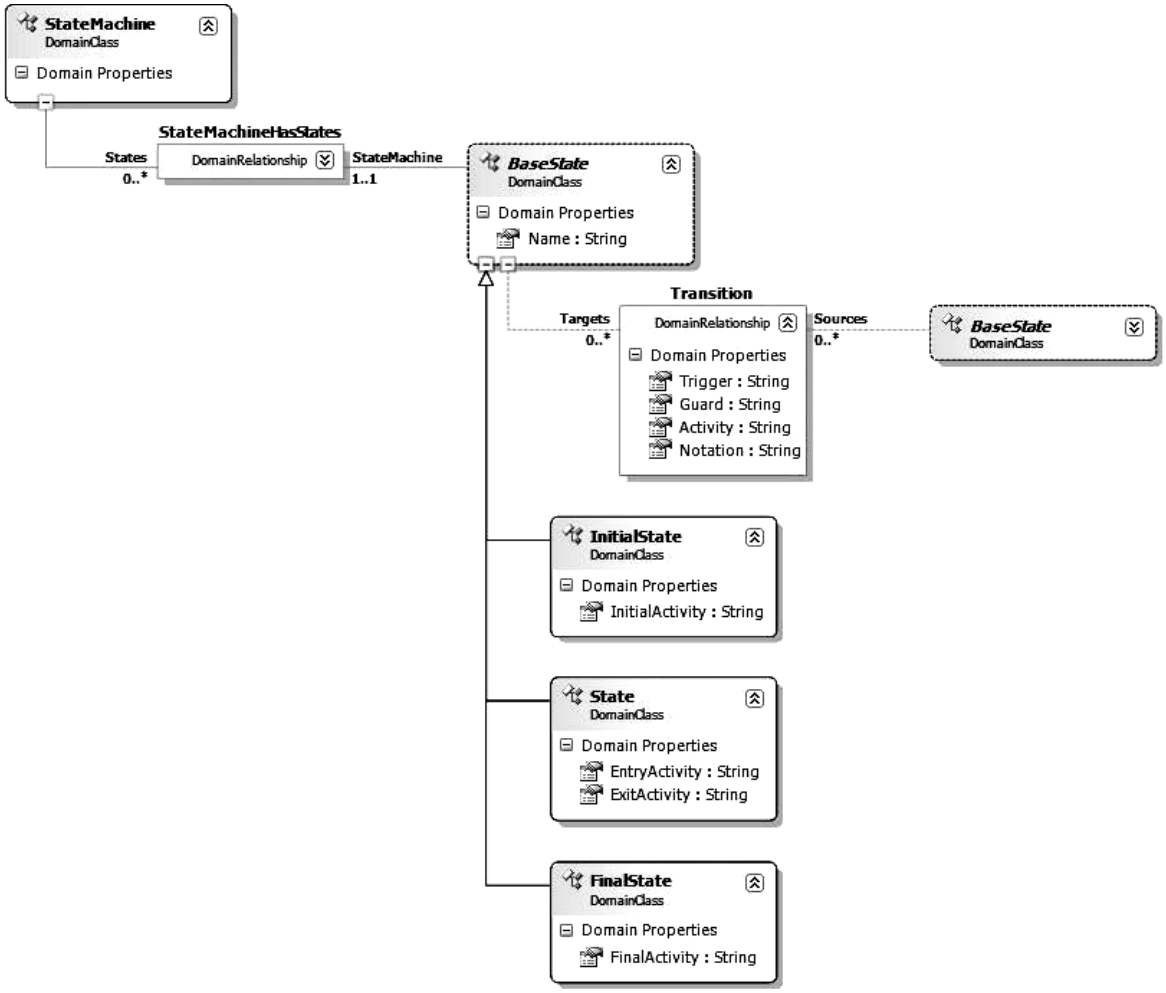

Рис. 5. Метамодель діаграми інтелектуального агента

Кореневим елементом метамоделі є сама діаграма інтелектуального агента [1] StateMachine. Відношення StateMachineHasStates означає, що агент перебуває у станах. BaseState – базовий тип для стану. На діаграмі можливе розміщення трьох типів станів: початковий стан InitialState, проміжний стан State і кінцевий стан FinalState. Будь-який проміжний стан може мати дві дії: EntryAction – дія, яку буде виконано відразу під час входу в цей стан, ExitAction – дія, яка буде виконана під час виходу зі стану. Початковий та кінцевий стани мають одиничну дію. Для початкового стану це дія InitialActivity. Вона буде виконана під час запуску агента. Для кінцевого стану це дія FinalActivity. Вона буде виконана по завершенню роботи інтелектуального агента.

Відношення Transition означає перехід з одного стану в інший.

Кожний перехід має назву події Trigger, під час виникнення якої буде здійснено перехід. Стрічка Guard подає собою умову, виконання якої необхідно для здійснення переходу. У випадку наявності та виконання необхідної умови, буде виконано дію Activity. Notation – стрічка, яка генерується автоматично і подає собою повний опис переходу в форматі Trigger[Guard]/Action.

Основним класом для роботи з джерелами даних є клас Model (специфікація наведена на рис. 6). Розроблений за вимогами [4].

Цей клас має такі властивості й методи:

- Connection – послання на з'єднання з джерелом;
- Description – опис джерела;
- FileName – назва файла, в якому знаходиться джерело;
- MetaModelName – назва метамоделі, на основі якої розроблено джерело (тип джерела);
- Models – список пов'язаних джерел;
- Name – назва джерела;
- Entities – колекція сутностей;
- Relations – колекція відношень;
- Load – метод, який відповідає за завантаження СДІР в каталог та словник даних;
- Modify – метод, що відповідає за зміну в каталозі та джерелі даних;
- RemoveModel – метод, що відповідає за знищення інформації про джерело.

Колекція сутностей джерела описується класом EntityCollection, специфікацію якого наведено на рис. 7.

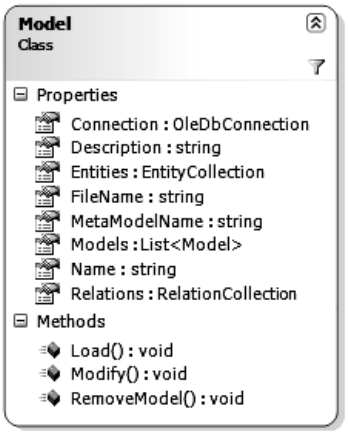

Рис. 6. Специфікація класу Model

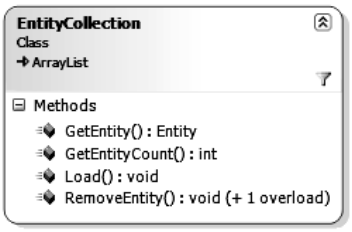

Рис. 7. Специфікація класу EntityCollection

Клас EntityCollection має такі методи:

- GetEntity – метод, який повертає елемент колекції сутності за іменем;
- GetEntityCount – метод, що повертає число екземплярів сутності, створених у поточній моделі;
- Load – метод, відповідальний за завантаження колекції сутностей поточної моделі із БД;
- RemoveEntity – метод, відповідальний за видалення сутності з колекції й БД.

На рис. 8 наведено опис класу Entity, що реалізує поняття "Сутність".

Клас Entity містить такі властивості й методи:

- Attributes – список атрибутів сутності;
- Constraints – список обмежень, що накладаються на сутність;
- Count – кількість екземплярів сутності, що можна створити в моделях;
- Description – опис сутності;
- Name – назва сутності;
- EntityDrawType – піктограма для відображення сутності;
- Entities – колекція сутностей моделі, якій належить поточна сутність;

• EntityType – тип сутності; для метамоделей це “Сутність”, а для моделей тип визначається сутностями метамодели;

• Operations – список припустимих над сутністю операцій;

• Values–- список значень атрибутів сутності;

• GetAllRelations – метод, що повертає список всіх відносин сутності;

• GetInRelations – метод, що повертає список відносин, для якої ця сутність є приймачем;

• GetOutRelations – метод, що повертає список відносин, для якої ця сутність є джерелом;

• Modify – метод, відповідальний за зміну сутності в БД;

• SaveToDataBase – метод, відповідальний за збереження сутності в БД.

Клас Relation реалізує поняття “Відношення” у моделях (рис. 9). Цей клас має такі властивості:

• Constraints – список обмежень, що накладаються на відношення;

• Description – опис відношення;

• EndEntity – посилання на сутність-приймач відношення;

• EndEntityMax – максимальна кількість екземплярів сутності-приймача відношення;

• EndEntitytMin – мінімальна кількість екземплярів сутності-приймача відношення;

• Name – назва відношення;

• Relations – колекція відношень, якій належить поточне відношення;

• StartEntity – посилання на сутність-джерело відношення;

• StartEntityMax – максимальна кількість екземплярів сутності-джерела відношення;

• StartEntityMin – мінімальна кількість екземплярів сутності-джерела відношення;

• Type – тип відношення.

Методом класу, відповідальним за збереження відношення в каталозі даних, є *SaveToDataBase*.

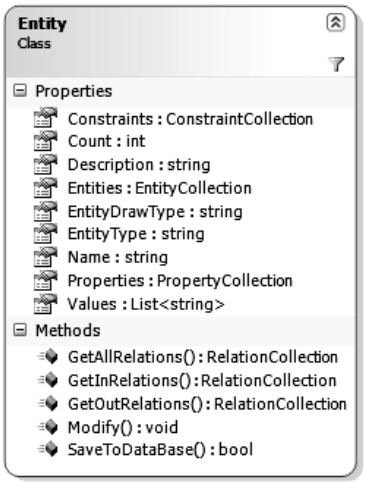

Рис. 8. Специфікація класу Entity

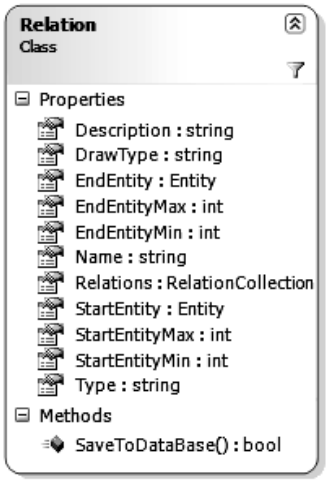

Рис. 9. Специфікація класу Relation

Клас RelationCollection описує колекцію відношень (див. рис. 10).

Клас RelationCollection має такі методи:

• GetRelation – метод, що повертає елемент колекції відношення за його назвою;

• Load – метод, відповідальний за завантаження всіх відношень поточного джерела даних;

• RemoveRelation – метод, відповідальний за видалення відношення з колекції й каталогу даних.

Поняття “Атрибут” описується класом Attribute. Специфікацію класу наведено на рис. 11 Властивостями й методами класу є:

• Default – значення за замовчуванням атрибута;

• Description – опис атрибута;

• Name – назва атрибута;

• Type – тип атрибута; тип може бути посиланням на домен припустимих значень або посиланням на деяку сутність;

• SaveToDataBase – метод, відповідальний за збереження атрибута в каталозі даних.

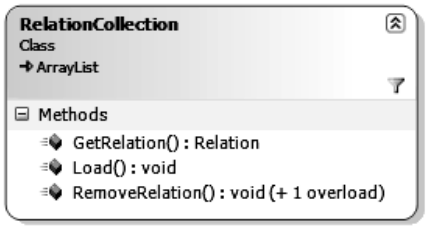

Рис. 10. Специфікація класу RelationCollection

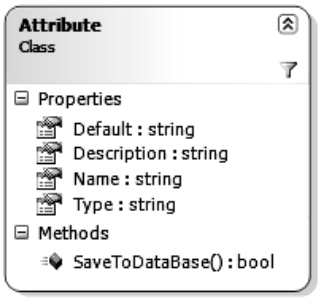

Рис. 11. Специфікація класу Attribute

Колекція атрибутів сутності задається класом AttributeCollection. Опис цього класу наведено на рис. 12.

Клас AttributeCollection задається такими методами:

• GetAttribute – метод, що повертає елемент колекції атрибутів за його назвою;

• Load – метод, відповідальний за завантаження колекції атрибутів поточної сутності із джерела;

• RemoveAttribute – метод, відповідальний за видалення атрибута з колекції й каталогу даних.

Клас, специфікацію якого наведено на рис. 13, описує поняття “Обмеження”.

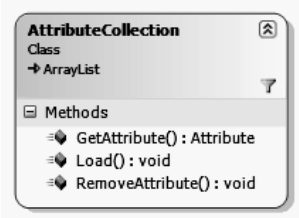

Рис. 12. Специфікація класу AttributeCollection

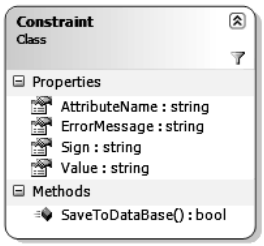

Рис. 13. Специфікація класу Constraint

Клас має такі властивості:

• ErrorMessage – текст повідомлення про помилку, що буде видано, якщо обмеження не виконується;

• AttributeName – назва атрибута сутності, на яке накладається обмеження;

• Sign – знак обмеження;

• Value – начення атрибута в правій частині обмеження.

Методом класу, відповідальним за збереження обмеження в БД, є SaveToDataBase.

Колекція обмежень, що накладаються на сутності й відношення, задається класом ConstraintCollection. Опис цього класу наведено на рис. 14.

Методами класу ConstraintCollection є:

• Load – метод, відповідальний за завантаження колекції обмежень;

• RemoveConstraint – метод, відповідальний за видалення обмеження з колекції й каталогу даних.

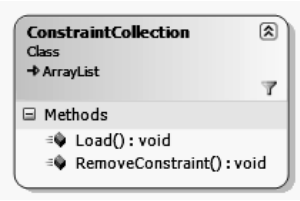

Рис. 14. Специфікація класу ConstraintCollection

7. Розроблення мовних засобів

Однією із задач, що виникають при побудові транслятора, є визначення вхідної метамови запиту до простору даних. Для цього використано розширену нотацію Бекуса–Наура (Backus/Naur Form – BNF).

<letter>::=a|b|c|d|e|f|g|h|i|j|k|l|n|m|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|N|M|O|P|Q|R|S|T|U|V|W|X|Y|Z
<keyword> ::= (<keyword>) |<letter> | < keyword>
<number> ::= 0|1|2|3|4|5|6|7|8|9
<object> ::= <елемент каталогу даних>
<par> ::= <синонім елемента каталогу даних>
<param> ::= <keyword>[{<keyword>|<number>}]
<num> ::= <number>[{<number>}]
<expr> ::= <operand> [{<op> <operand>}]
<operand> ::=" (“<expr>”)” | <num> | <param> [“[“<expr>”]”]
<op> ::= <grteq>
<inv> ::= <logicalop> | “*” | “/”
<type> ::= “SUM” | “COUNT” | “AVG”
<logicalop>::= “<” | “>” | “>=” | “<=” | “=” | “<>” | [<op>]
<whereop> ::= “where” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] }“)”
<whoop> ::= “who” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”
<howop> ::= “how” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”
<Seop> ::= “Se” “(”<object>[“:”<par>] [“Agg”<type>] {“,” <object> [“:” <par>] [“Agg” <type>] }“)”
<whatop> ::= “what” “(” <object> [“:” <par>] [“,”<object> [“:” <par>]]“)”
<whichop>::= “which” “(” <object> [“:” <par>] [“,”<object> [“:” <par>]]“)”
<Semantop>::= “Semant” “(” <object> [“,” <object>]“)”
<Consop> ::= “Cons” “(”<object> [“:” { <par> <operator> <param>}] “)”
<profileop> ::= “where” “(” <object> [“:” <num>] {“,”<object> [“:” <num>] }“)”
<Unionop> ::= “Union” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”
<Unionop> ::= “Union” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”
<Intersop> ::= “Inters” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”
< Differop> ::= “Differ” “(” <object> [“:” <par>] {“,”<object> [“:” <par>] } “)”

8. Розроблення інтерфейсу

Нижче показано метамодель інтерфейсу формування запиту користувача (рис. 15). Відношення InterfaceHasMethods, InterfaceHasProperties, InterfaceHasEvents означають, що інтерфейс може мати Методи, Властивості і Події.

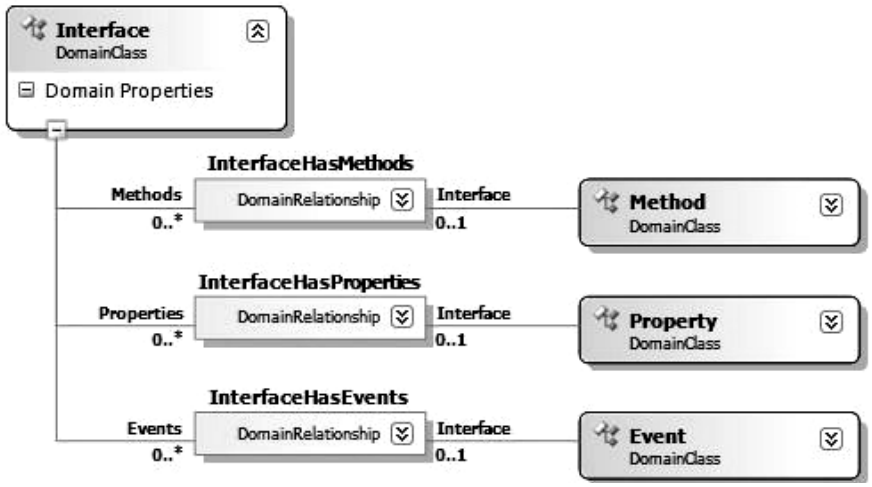

Рис. 15. Метамодель інтерфейсу формування запиту користувача

Для розроблення порталу як архітектурний шаблон використовували шаблон Model-View-Controller (MVC).

Модель-Вид-Контролер (Model-View-Controller, MVC) – архітектурний шаблон (рис. 16), який використовується для проектування та розроблення програмного забезпечення. Поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відділення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли проводитися без зміни інтерфейсу користувача.

Мета шаблону – гнучкий дизайн програмного забезпечення, який має полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонент програми. Крім того, використання цього шаблону в великих системах приводить їх в певний порядок і робить зрозумілішими завдяки зменшенню їх складності.

Архітектурний шаблон Модель-Вид-Контролер (MVC) поділяє програму на три частини. У тріаді до обов'язків компонента Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідає за представлення цих даних користувачеві. Контролер (Controller) управляє компонентами, отримуючи сигнали у вигляді реакції на дії користувача і повідомляючи про зміни Модель-компоненту.

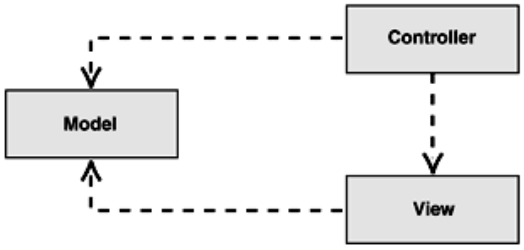

Рис. 16. Архітектура програми, реалізованої з використання з використанням шаблону MVC

Модель інкапсулює ядро даних і основний функціонал з їхньої обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу Вигляд може мати декілька взаємозв'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. До функцій Контролера належить моніторинг за подіями, що виникають у результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле). Реєстровані події транслюються в різні запити, що скеровуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Поділ моделі від представлення даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Тобто, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, представлена одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

На рівні Model використовується ORM (Object-relational mapping), зокрема технологія Entity Framework. На цьому рівні створюється модель БД, що дає змогу працювати з нею як з набором сутностей, а також уникнути явного використання SQL. Усе це виконає ORM.

Controller являє собою клас, який містить обробники подій та іншу бізнес-логіку.

Висновки

Спроектовано архітектуру простору даних та інструментальних засобів, що стало основою для практичної реалізації. Обрано проектне рішення для реалізації засобів інтеграції даних з різнотипних джерел. Розроблено специфікації основних класів. Описано мовні засоби та принципи реалізації інтерфейсу користувача.

1. Шаховська Н. Б. Програмне та алгоритмічне забезпечення сховищ та просторів даних: монографія [Текст] / Н.Б. Шаховська; – Львів: Видавництво Львівської політехніки, 2010. – 194 с. 2. Шаховська Н.Б. Методи опрацювання консолідованих даних за допомогою просторів даних / Н.Б. Шаховська // Проблеми програмування / Національна академія наук України, Інститут програмних систем НАН України. – 2011. – № 4. – С. 72–84. 3. Матов О. Я. Сучасні технології інтеграції інформаційних ресурсів / О. Я. Матов, І. О. Храмова // Реєстрація, зберігання і обробка даних. – 2009. – Т. 11, № 1. – С. 33–42. 4. The Open Archives Initiative Protocol for Metadata Harvesting Protocol Version 2.0 of 2002-06-14. – [Електронний ресурс]. – [Режим доступу] http://www.openarchives.org /OAI/2.0/openarchivesprotocol.htm