УДК 004.852

Магас Валентин Васильович
студент
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Магас Валентин Васильевич
студент
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Mahas Valentyn
Student of the
National Technical University of Ukraine
«Igor Sikorsky Kyiv Polytechnic Institute»

КОМПОНУВАННЯ СЕМАНТИЧНИХ REST-СЕРВІСІВ З ВИКОРИСТАННЯМ NEO4J

КОМПОНОВКА СЕМАНТИЧЕСКИХ REST-СЕРВИСОВ С ИСПОЛЬЗОВАНИЕМ NEO4J

COMPOSITION OF SEMANTIC REST-SERVICES USING NEO4J

Анотація. *Пропонується модель опису сервісів, що орієнтована на гіпермедіа, яка передбачає генерування графа, що захоплює переходи станів (операції HTTP) в шарі активності. Модель сервісів була реалізована у вигляді анотацій і опису JSON. Прототип був розроблений з використанням Neo4j; набір реальних Web API був обраний для ілюстрації нашого підходу.*

Ключові слова: *Microdata, Neo4j, REST-сервіс, JSON.*

Аннотация. *Предлагается модель описания сервисов, ориентированная на гипермедиа, которая предусматривает генерирование графа, захватывает переходы состояний (операции HTTP) в слое активности. Модель сервисов была реализована в виде аннотаций на и описания JSON. Прототип был разработан с использованием Neo4j; набор реальных Web API был выбран для иллюстрации нашего подхода.*

Ключевые слова: *Microdata, Neo4j, REST-сервис, JSON.*

Summary. *We offer a service description model focused on hypermedia that allows the generation of a graph that captures state transitions (i.e. HTTP operations) in an activity layer. The service model was implemented as annotations, and a JSON description. A prototype was developed using Neo4J, and a set of real Web APIs was chosen to illustrate our approach.*

Key words: *Microdata, Neo4j, REST-service, JSON.*

Вступ. Існує два основні підходи до створення Web-сервісів. Один заснований на WSDL / SOAP і більшою мірою поширений в сценаріях B2B; інший — REST-сервіси або Web API (Application Programmable Interfaces), може часто застосовуються у Web. Web API — це популярний_спосіб надання доступу до ресурсу, уникаючи складності стека технологій і стандартів, чого вимагають сервіси створені на основі SOAP. На практиці Web API має певні недоліки, такі як виклики RPC тунельовані протоколом HTTP і обмежена еволюційність сервісу, оскільки зв'язок між клієнтом і сервером відбувається шляхом опису сервісу, написаного природною мовою і, зазвичай, надається у вигляді HTML-сторінок. REST сервіси, на противагу, вимагають додаткових обмежень, таких як узгодження контенту, відповідне використання мережевого протоколу і гіпермедіа.

REST-сервіси часто супроводжуються неофіційними документами(наприклад, HTML-сторінки), які написані природною мовою, і використання інтерфейсу сервісу машинними клієнтами. Наразі, немає універсального вирішення даної проблеми і дієвою практикою вважається підключення бібліотек, які додають опис до сервісів.[1] Такий підхід часто неточний, і змушує розробників машинного клієнта брати участь у фазі пробної помилки. У цих умовах, автоматичне виявлення та компонування сервісів важке для підтримки або і зовсім не можливе для реалізації. Основна складність полягає в тому, що в поточних описах сервісів необхідний людський інтелект щоб зрозуміти очікування сервісу від клієнта.

У нашій роботі пропонується підхід, який є сервісним описом, що враховує уніфіковане обмеження інтерфейсу REST, тобто, ідентифікація ресурсів, маніпулювання ресурсами через представлення та гіпермедіа як двигун стану програми. Наведемо реалізацю в JSON, що дозволяє створювати зручну для читання документацію.

Реалізація заснована на метамоделі (Рисунок 1), яка передбачає існування двох шарів: *активності* (охоплює механізм зміни стану ресурсу) і *семантики* (охоплює семантику ресурсу).

Щоб проілюструвати наш підхід, розглянемо наступний сценарій, що формується з трьох веб-інтерфейсів: Spotify (https://developer.spotify.com/web-api/), Songkick (https://www.songkick.com/developer/) і Uber (https://developer.uber.com/). API Spotify надає доступ до каталогу служби потокової передачі музики. API Songkick надає доступ до бази даних живої музики з інформацією про майбутні і минулі концерти, а також сетлісти. Uber API дозволяє користувачеві отримувати інформацію про типи транспортних послуг, ціну і передбачуваний час прибуття, а також профіль і діяльність користувача.

Розглянемо прикладну задачу композиції сервісів: *Користувач хоче відвідати концерт його улюбленого гурту але найближчим часом концертів не передбачається. Тому він вирішує сходити на концерт схожого виконавця. Для цього йому потрібно дізнатися конкретне місце проведення, а також деталі поїздки на таксі на обрану подію.*

Така ціль може бути досягнута операціями GET (інформація про виконавців, концертні дані, ціни тощо) до ресурсів Songkick або Spotify. Взаємодія з Uber потрібна консультацій стосовно тарифів (GET), а потім для виклику таксі. Характерно, що інтерфейси неоднорідні і семантика сервісу дозволить виявити відповідні ресурси. Крім того, необхідно опрацьовувати різні ситуації (наприклад, гурт не підходить по стилю, місце події може бути занадто далеко, або оплата за таксі може бути занадто високою). Також важливо розглянути семантику операцій, оскільки ресурс може включати кілька посилань, що відповідають різним запитам з різною семантикою.

Реалізація опису сервісів у JSON

Перейдемо до знайомства з використанням JSON для створення окремого і повного опис RESTful веб-сервісу. Він задовольняє дві вимоги: генерування легкої для читання документації і надання машинним клієнтам правил взаємодії з сервісом. Ознайомитись з реалізацією можна на рисунку 2.

REST-сервіс описаний за допомогою таких елементів як *name, description, base URI* і *version*. Як видно з рисунку 3, *prefix* являє собою простір імен ключових значень (починаючи з символу «@») що скорочує посилання на семантичні елементи. На відміну від моделі Microdata, модель JSON описує набір ресурсів, отже, веб-сервіси складаються з ресурсів (*Rescourses*), описаних в зручному для сприйняття

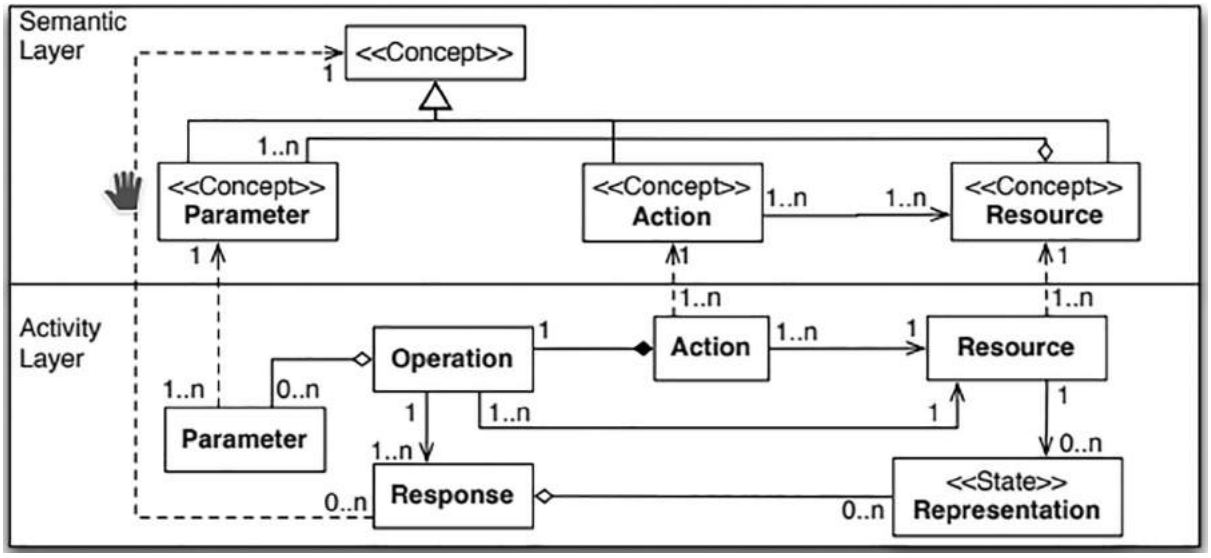

Рис. 1. Метамодель

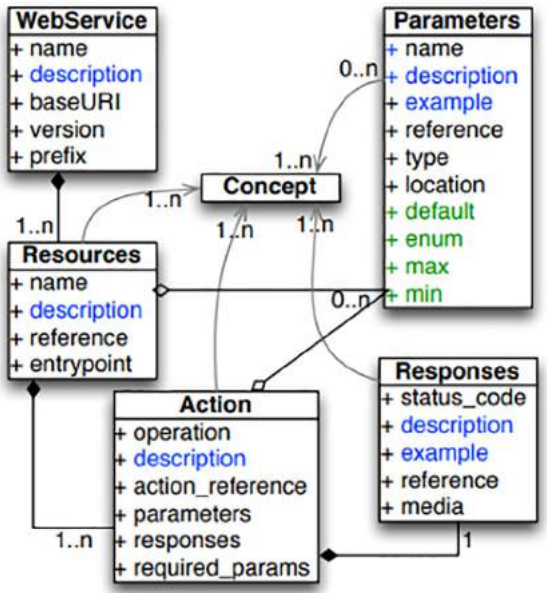

Рис. 2. Реалізація метамоделі з використанням JSON

людиною форматі, а також концептуальний об'єкт, який представляє його семантику через посилання (URI). Ресурси є наріжним каменем нашого опису. Кожен ресурс має точку входу, деякі точки входу є абсолютними URL-адресами, тоді як інші посилаються на екземпляри ресурсів, і, як наслідок, URL-адрес включає параметри, значення яких повинні бути визначені під час виконання (type=»path»); такі параметри також посилаються на концепцію. Ресурси, назва та опис в цілому також посилаються на відповідну концепцію ресурсів. Дії пов'язані з операціями, тобто з мережевим протокольним методоми, що застосовуються до даного ресурсу. Кожна операція має посилання на концепцію дії, яка визначає операцію на рівні бізнесу (наприклад для покупки, оренди, пошуку тощо).

Операції також мають параметри і логічний вираз, які дозволяють розробникам вказувати правила, що визначають, які параметри є обов'язковими. Параметри мають ім'я, тип (bool, string), місцезнаходження *(path, url, header* або *body*), посилання на концепцію і додаткові специфікації в залежності


```
{
"name": "Songkick",
"baseURI": "http://api.songkick.com/api/3.0",
"version": "3.0",
"description": "The Songkick API gives you easy access to the biggest live music ....",
"prefixes": {
  "@schema": "http://schema.org",
  "@Thing": "@schema/Thing",
  "@Action": "@schema/Action",
  "@apikey": "@Thing/CreativeWork/SoftwareApplication/WebApplication/apikey/songkickApiKey",
  "@identifier": "@Thing/identifier",
  "@artist_id": "@identifier$Organization/PerformingGroup/MusicGroup/songkickId",
  "@Collection": "@Thing/Collection",
  ...
},
"resources": {
   "/artists/{@artist_id}/similar_artists.json": {
      "name": "Collection of similar artists",
      "reference": "@Collection/MusicGroupCollection",
      "description": "A list of artists similar to a given artist, based on our tracking and ....",
      "actions": {
         "get": {
            "description": "Get a list of artists similar to a given artist.",
            "reference": "@Action/SearchAction/SearchMusicGroupAction",
            "additional_doc": "http://www.songkick.com/developer/similar-artists",
            "required_params": "apikey",
            "parameters": {
              "apikey": {
                "name": "API Key",
                "description": "Your API Key",
                "reference": "@apikey",
                "type": "string",
                "example": "ABC123DEFG"
              }
            },
            "responses": {
                "200": {
                   "description": "On success, the HTTP ...",
                   "media": [
                      "application/json"
                   ],
                   "reference": [
                      "@Collection/MusicGroupCollection"
                   ]
..
```


Рис. 3. Фрагмент, який зображає використання JSON для опису ресурсу схожих виконавців

від кожного типу (*maximum, minimum, enumeration* і *default*). Нарешті, відповідь (*Response*) повністю з'єднує граф, що містить посилання на концепцію, яка очікується для повернення разом з кодом стану HTTP та описом. Природно, що ми описуємо характеристики очікуваної відповіді, однак через динамічний характер REST фактична відповідь може змінюватися.

Композиція семантичних REST сервісів

Інформація про веб-сервіси (вершини та ребра) зберігається у графовій базі даних Neo4j. Ця база даних була обрана через відчутну гнучкість, продуктивність і хорошу масштабованість. Для поточної реалізації входом є JSON. Було релізовано два Python парсери. Перший перетворює HTML описи в JSON. Другий обробляє опис JSON і генерує вершини та ребра, які будуть зберігатися в базі даних з використанням бібліотеки Py2neo. JSON описи перевіряються за допомогою JSON Schema перед її парсингом.

Описи сервісів, аналізуються для створення графу (рисунок 4). Вершини та ребра позначені атрибутами: закруглені прямокутники для вершин і прямокутні прямокутники для ребер. Вершини та ребра мають внутрішній ідентифікатор в графі, *GRI* (ідентифікатор ресурсу графа). *Resources*, *Operations*, *Parameters* і *Responses* — це вершини в графі, тоді як *Actions* стають семантичним ребром, що з'єднує ресурси і операції. *Resources, Operations, Parameters* та *Actions* пов'язані з *Concepts* ребром *reference*.

Concepts самі є вершинами, описаними двома атрибутами, *URI* і *label*, що позначає його тип. *Concepts* можуть відповідати складній семантичній моделі, такій як онтологія (наприклад, відношення «isA» від вершини 6 до 3). У випадку параметрів вони класифікуються на 4 типи (*path, url, header* і *body*), і відносяться до Concept *Parameter*. Відносини між шарами активності RAD також включені і анотуються за допомогою семантики відносини та ідентифікатора *GRI*.

Повернемося до сформульованої вище задачі з композиції семантичних сервісів. Враховуючи, що користувач знає словник Schema.org, йому потрібно виконати наступні запити:

1. По-перше, необхідно виявити сервіси, які дозволяють користувачеві шукати аналогічних виконавців (action.GRI = «/GetSimilarArtistsAction»), що забезпечують *MusicGroupCollection* як результат. На рисунку 5 (b) показаний запит Cypher, який вирішує поставлене завдання.

2. Згодом користувач повинен знайти інформацію про виконавців, отриманих в попередньому запиті (рисунок 5 (c)). У цьому випадку тільки Songkick може забезпечити таку функціональність, як показано на рисунку 6 (b).

3. Наступний крок — знайти інформацію про доступність таксі для знайденого місця проведення (рисунок 5 (d)). Тепер тільки один сервіс (Uber) надає точку входу такий запит: «/GetTaxiCollectionAction» на рисунку 6 (c). Запити (e) і (f) на рисунку 5 представляють наступні кроки: знайти ціни і прогнозований час у дорозі і надати відповіді, аналогічні запитом для доступності таксі, як показано на малюнку 6 (d).

Зверніть увагу, що на даний момент композиція сервісів не може виконуватися автоматично, так як необхідно підтримувати потік даних. Наприклад, для знаходження схожих виконавців, користувач повинен надати інформацію, таку як *artists_id* і *apikey* для випадку Songkick і назва музичної групи для випадку Spotify. Тільки два ресурси зі знайдених 7 підтримують параметри *artists_id* і *apikey*, Artist's Gigography і ресурс Calendar by Artist, обидва вимагають додаткові параметри конфігурації, такі як *page*, *order* і *per_page*. Однак семантика попередньо-

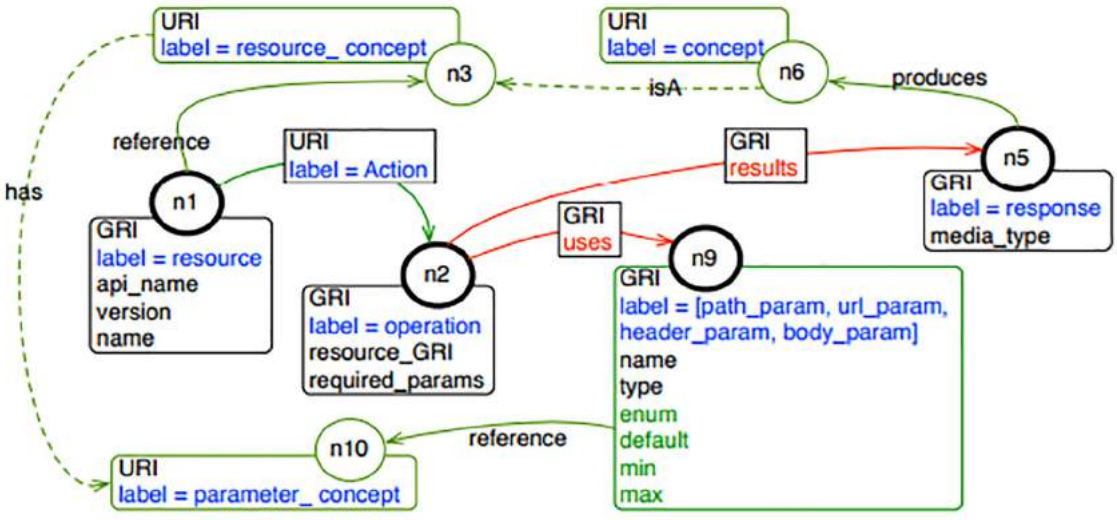

Рис. 4. Структура графа реалізації підходу

го ресурсу відноситься до минулих подій, тоді як останній ресурс відноситься до майбутніх подій. У всіх цих випадках необхідне втручання людини.

Висновок. В даній роботі було сформовано два принципових припущення: розробники сервісів будуть дотримуватися загальної лексики, щоб семантично анотувати їх сервіси і що відповіді сервісів слідують підходу гіпермедіа, що з'єднує їх через посилання. Були запропоновані різні методи для автоматичного зв'язування ресурсів і концепцій [7]. На основі методів, реалізовано робочий прототип, що проілюстрував життєздатність зроблених припущень. Ми слідували евристичному та конвенційному підходам, оскільки реалізуємо пробний інструмент, проте наш намір полягає в продовженні вивчення обробки природної мови і non-logical підходу.

Наш підхід додає новий рівень до архітектури REST, що становить певну проблему для обслуговування. Також важливою проблемою є потік даних між операціями, тобто, перетворення даних відповіді в параметри для виконання нової операції. Завдання не тільки передбачає використання різних типів даних (string, boolean, integer), але також і створення даних, які семантично еквівалентні (хеш-коди для різних API, які виробляються слідуючи різним протоколам).

Отже, анотації виглядають ефективною альтернативою для досягнення поставленої мети. Ця робоча пропозиція спрямована на надання стратегії для виявлення і компонування REST-сервісів, які включають опис, зрозумілий як розробникам так і машинам. В якості подальшої роботи ми зосередимося на виявленні зв'язків між ресурсами на семантичному рівні, та рівні активності.

(a)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters)
WHERE concept.GRI=~ 'http://schema.org/Thing/CreativeWork/MusicPlaylist.*'
AND action.GRI = 'http://schema.org/Action/GetAction/GetMusicRecordingCollectionAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter
RETURN concept, resources, operations, parameters
```

(b)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/MusicGroupCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetMusicGroupCollectionAction/GetSimilarArtistsAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter         AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(c)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/MusicEventCollection'
AND action.GRI = 'http://schema.org/Action/SearchAction/MusicEventSearchAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter         AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(d)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiCollectionAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter         AND responses: Response
RETURN concept, resources, operations, responses, parameters
```

(e)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiFareCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiFareCollectionAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter         AND responses:Response
RETURN concept, resources, operations, responses, parameters
```

(f)
```
MATCH (concept)<-[:`is a`]-(resources)-[action]->(operations)-[:uses]->(parameters), (operations)-[:results]-(responses)
WHERE concept.GRI = 'http://schema.org/Thing/Collection/TaxiTravelTimeCollection'
AND action.GRI = 'http://schema.org/Action/GetAction/GetTaxiTravelTimeCollectionAction'
AND concept: `Resource Concept`   AND resources: Resource   AND operations: Operation
AND parameters: Parameter         AND responses:Response
RETURN concept, resources, operations, responses, parameters
```

Рис. 5. Запити Cypher

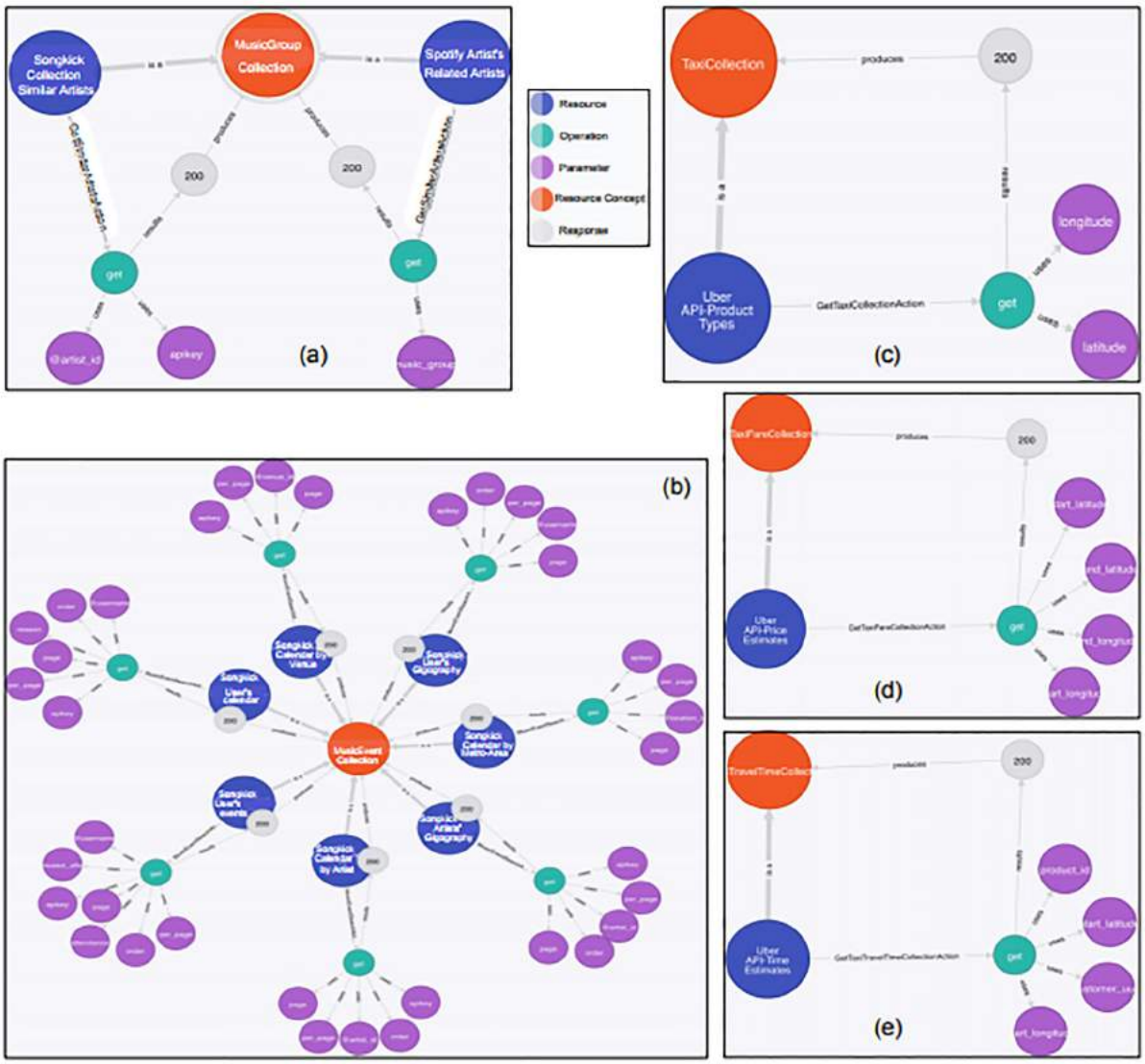

Рис. 6. Результати виконання запитів з рисунку 5

Література

1. S. Mayer and D. Guinard, «An extensible discovery service for smart things,» in Proceedings of the Second International Workshop on Web of Things. ACM, 2011, p. 7.

2. R. T. Fielding, «Architectural styles and the design of network-based software architectures,» Ph.D. dissertation, University of California, Irvine, Irvine, California, 2000.

3. R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, and S. Weerawarana, «Web services description language (wsdl) version 2.0 part 1: Core language,» W3C working draft, vol. 26, 2004.

4. M. Hadley, «Web application description language,» World Wide Web Consortium, Member Submission SUBM-wadl-20090831, August 2009.

5. D. John and M. Rajasree, «Restdoc: Describe, discover and compose restful semantic web services using annotated documentations,» International Journal of Web & Semantic Technology (IJWesT), vol. 4, no. 1, 2013.

6. R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. Gabarr´o Vall´es, «Description and interaction of restful services for automatic discovery and execution,» in Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services, 2011.

7. J. Lathem, K. Gomadam, and A. P. Sheth, «Sa-rest and (s)mashups: Adding semantics to restful services,» in First IEEE International Conference on Semantic Computing (ICSC2007), Irvine, California, September 2007, pp. 469–476.

8. J. Kopeck´y, K. Gomadam, and T. Vitvar, «hrests: An html microformat for describing restful web services,» in 2008 IEEE/WIC/ACM International Conference on Web Intelligence, Sydney, Australia, December 2008, pp. 619–625.

9. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton, «Rdfa in xhtml: Syntax and processing — a collection of attributes and processing rules for extending xhtml to support rdf,» World Wide Web Consortium, Recommendation REC-rdfa-syntax- 20081014, October 2008.