УДК 004.852

Кравчук Євгеній Сергійович
студент
Національний технічний університет України «Київський політехнічний інститут»
Кравчук Евгений Сергеевич
студент
Национальный технический университет Украины «Киевский политехнический институт»
Kravchuk Y.
student
National Technical University of Ukraine «Kyiv Polytechnic Institute»

АНАЛІЗ ПРОТОКОЛІВ ВЗАЄМОДІЇ У КЛІЄНТ-СЕРВЕРНІЙ АРХІТЕКТУРІ

АНАЛИЗ ПРОТОКОЛОВ ВЗАИМОДЕЙСТВИЯ В КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЕ

ANALYSIS OF COMMUNICATION PROTOCOLS IN CLIENT-SERVER ARCHITECTURE

Анотація. *Метою даної роботи є аналіз існуючих протоколів взаємодії веб-сервісів з поадльшим виявленням переваг та недоліків кожного з протоколів та опис найбільш пасуючих випадків використання. Дана робота зосереджується як на будові кожного протоколу, так і на порівнянні його з іншими протоколами для ефективного аналізу.*

Ключові слова: *веб-сервіс, протокол взаємодії, SOAP, REST, RPC, XML-RPC.*

Анотация. *Целью данной работы является анализ существующих протоколов взаимодействия веб-сервисов с последующим выявлением преимуществ и недостатков каждого из протоколов и описание наиболее подходящих случаев использования. Данная работа сосредоточена как на строении каждого протокола, так и на сравнении его с другими протоколами для эффективного анализа.*

Ключевые слова: *веб-сервис, протокол взаимодействия, SOAP, REST, RPC, XML-RPC.*

Summary. *The aim of this research is the analysis of existing web-service communication porotcols with subsequent detection of advantages and disadvantages of each of the protocols and the description of the most suitable use cases. This study focuses on both structure of each protocol and comparison with other protocols to achieve effective analysis.*

Key words: *web-service, communication protocol, SOAP, REST, RPC, XML-RPC.*

Вступ

Сучасній мережі Інтернет характерна різнорідність додатків, що функціонують на різних вузлах мережі, апаратно-програмних платформах та використовують різноманітні технології та мови програмування. Якщо гіпотетично поділити Інтернет на декілька шарів, то виділяються щонайменше два концептуальних типа додатків — обчислювальні вузли, котрі реалізують нетривіальні функції, і прикладні веб-ресурси. В цей же час, другі частіше за все використовують послуги, що надають перші.

Для того, щоб зв'язати вищеописані сутності і надати одним додаткам можливість обмінюватися даними з іншими, були придумані веб-сервіси. Веб-сервіси по своїй суті — це реалізація абсолютно чітких інтерфейсів обміну даними між різними додатками, котрі написані не тільки різними мовами, але й розподілені на різних вузлах мережі.

Саме з появою веб-сервісів розвинулася ідея Service Oriented Architecture — сервіс-орієнтованої архітектури веб-додатків. Веб-сервіси виступають серверами, постачальниками послуг, для інших сервісів, котрі в цей час виступають клієнтами або споживачами.

Мета роботи

З огляду на ріст популярності веб-сервісів та сервіс-орієнтованої архітектури загалом, зростає важливість вибору протоколів взаємодії. Метою даною роботи є вибір найбільш доцільного протоколу взаємодії веб-сервісів для проектів різного маштабу та складності серед існуючих популярних протоколів, виявлення їх переваг та недоліків загалом та у контексті конкретних типів додатків.

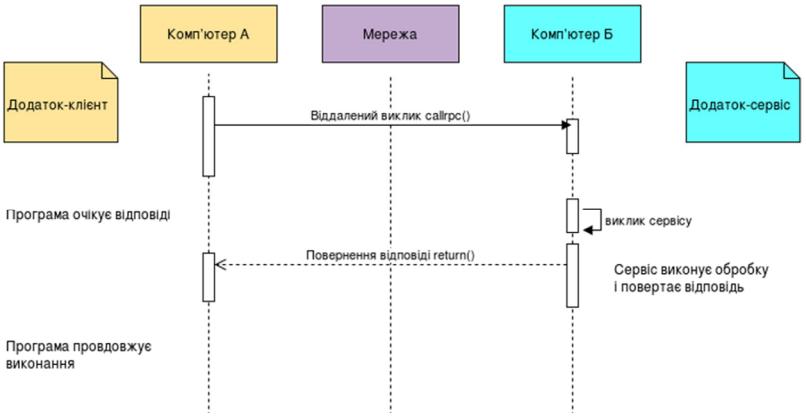

Рисунок 1. Діаграма послідовності RPC

Існуючі протоколи взаємодії веб-сервісів

На сьогоднішній день найбільш широке розповсюдження отримали SOAP, XML-RPC та REST. Вони відрізняються як контекстом, так і використанням. У світі XML існує два основних шляхи реалізації віддаленого виклику процедур: XML-RPC та SOAP. XML-RPC — протокол, котрий є пращуром SOAP і виділяється виключною простотою в застосуванні та використанні. SOAP — це протокол, в той час як REST — це архітектурний стиль. SOAP є добре продуманим протоколом, що використовується в веб індустрії і стандартизується World Wide Web Consortiumом (W3C). REST в свою чергу — це результат дисертації «Architectural Styles and the Design of Network-Based Software Architecture» 2000-го року, автором якої є Рой Філдинг. REST набирає популярності через свою простоту, масштабованість і архітектурну залежноість від World Wide Web. Великі корпорації, як-то Google і Amazon, зокрема, використовують REST у переважній більшості своїх продуктів. Найголовнішою відмінністю між цими моделями є те, що SOAP є сильнозв'язаною системою, водночас REST — слабозв'язана система.

RPC. XML-RPC

RPC (XML Remote Procedure Call) — віддалений виклик процедур з допомогою XML. Сама методика віддаленого виклику відома давно і використовується в таких технологіях як CORBA, DCOM, SOAP. RPC слугує для побудови розподілених клієнт-серверних обчислень.

На дуже спрощеному прикладі механізм роботи виглядає наступним чином: додаток, виконучи обробку деяких даних на локальному комп'ютері звертається до деякої процедури. Якщо її реалізація присутня в програмі, то процедура приймає параметри, виконує дію і повертає деякі дані. Якщо це віддалений виклик, то треба знати де буде виконуватися процедура.

Діаграма послідовності на рисунку 1 описує процес віддаленого виклику процедури між програмами на різних комп'ютерах у мережі.

Запит на виконання процедури разом з параметрами записується у вигляді XML-документа і передається по мережі посередництвом HTTP на інший комп'ютер, де з XML-документа вилучається ім'я процедури, параметри та інша потрібна інформація. Після завершення роботи процедури формується відповідь, і вона передається комп'ютеру, що надіслав запит.

Але формат обміну даними при класичній моделі RPC залишається бінарним, що означає складність у роботі з ним, якщо, нарпиклад, потрібно організувати роботу розподіленої системи, де між окремими ділянками мережі сотять фаєрволи та проксі-сервери. До того ж різні платформи мають свої тонкощі у використанні.

Ця проблема передувала створенню похідної технології XML-RPC компанією UserLand Software Inc. Передача даних у них виконується протоколом HTTP, формат даних — XML. Це знімає обмеження, накладені на конфігурацію мережі, так і на маршут слідування пакетів, — виклики XML-RPC являють собою простий тип даних text/xml і вільно проходять через шлюзи там, де допускається ретрансляція HTTP-трафіку.

Повідомлення XML-RPC передаються методом POST протокола HTTP. Вони бувають трьох типів: запит, відповідь і повідомлення про помилку. Протокол передбачає сім простих типів даних і два складних, для передачі параметрів методу і повертаємих значень.

Більш складні типи даних, наприклад об'єкти, треба передавати у бінарному вигляді чи заміняти структурами. Не дивлячись на те, що протокол є дещо застарілим, але він ще й досі використовується і має певні переваги та недоліки.

Таблиця 1

Основні характеристики технології XML-RPC

Характеристика	XML-RPC
Скалярні типи даних	Є
Структури	Є
Масиви	Є
Іменовані масиви та структури	Відсутні
Кодування, визначені розробником	Відсутні
Типи даних, визначені розробником	Відсутні
Деталізація помилок	Є
Легкість практичного застосування та засвоєння	Легкий для розуміння і практичного використання

SOAP

Це протокол для обміну веб-сервісами структурованою інформацією в комп'ютерних мережах. Мова для опису веб-сервісів WSDL (Web Service Description Language) використовується разом з SOAP, щоб зробити повідомлення доступними у Web посередництвом веб-сервісів. Загалом, SOAP сумісно з WSDL називається SOAP Web services.

Без протокола SOAP веб-сервіси не можуть працювати кросплатформенно, і зазвичай несумісні. SOAP був розроблений для заповнення цієї прогалини, полегшуючи розробнику конструювання веб-програми незалежно від вищезгаданих обмежень. SOAP в данний момент залежить від HTTP, що лежить в основі, але може використовувати інші протоколи передачі даних. Microsoft були першими, хто почав розроблювати SOAP, а далі й інші відомі корпорації почали вносити свій вклад у розробку.

Раніше в історії персональних комп'ютерів, базова обробка даних складалася з одного персонального комп'ютера з використанням простих додатків, що залежали від локальних ресурсів. Розробки у сфері комп'ютерних мереж поступово популяризували концепції комп'ютерних мереж LAN, MAN та WAN, даючи можливість спілкуватися комп'ютерам, що фізично знаходяться у різних місцях.

Протоколи веб-сервісів дали можливість комп'ютерам взаємодіяти на однакових платформах, технологіях та операційних системах. Потреба у протоколі, незалежному від цих обмежень породила протокол SOAP, котрий використовує XML для спілкування поверх транспортного рівня.

Призначення SOAP-а — надати мінімальний транспортний функціонал, поверх якого можуть бути побудовані більш складні протоколи та процеси взаємодії.

Для розробки клієнта треба знати URL кінцевої точки сервера, яка відповідає за обробку повідомлень. Це стає ще легшим з використанням WSDL, форматом XML схеми, який надає можливість представити мережеві сервіси у вигляді множини кінцевих точок, що обробляють повідомлення.

Офіційно протокол був стандартизований консорціумом W3C у 2003 році, використовується для представлення бізнес даних через інтерфейси і сервіси, що обмінюються цілими документами чи відображають дані на об'єкт, використовуючи назви методів, вхідні та вихідні параметри. Наступний уривок коду є прикладом типового SOAP повідомлення:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 299
<?xml version="1.0">

<soap: Envelope xmlns: soap="http://w3.org/2003/05/
soap-envelope">
        <soap: Header>
        </soap: Header>
                <soap: Body>
                        <m: GetStockPrice xmlns: m=
"http://www.example.org/stock">
                                <m: StockName>IBM
</m: StockName>
                        </m: GetStockPrice>
                </soap: Body>
</soap: Envelope>
```

Використані елементи мають наступні значення:

- *Envelope* — є індикатором початку та кінця повідомлення;
- *Header* — містить опціональні допоміжні дані для обробки повідомлень;
- *Body* — складається з головних XML даних;

Щоб бути незалежними від HTTP, SOAP має перевагу, а саме, елемент *attachment* — надсилання прикріплених файлів, що часто вважають найкращим рішенням для сервісів, що мають справу з прикріпленими файлами.

Через свою надійність та рівень безпеки протоколи SOAP широко використовуються у банківських системах та корпоративних додатках.

REST

REST — це архітектура для розробки веб-сервісів, що намагається імітувати архітектури, котрі використовують HTTP чи схожі протоколи, шляхом обмеження інтерфейсів набором стандартних і добре знайомих розробникам операцій (наприклад, GET, PUT/PATCH, POST та DELETE для HTTP).

Акцент зроблений на взаємодії з ресурсами, що мають збережений стан, а не з повідомленнями та

операціями. Можна сказати, що ця архітектура розроблена для того, щоб показати, що існуючого HTTP достатньо для побудови веб-сервісу, та демонстрації маштабування.

Рой Філдинг, один із основних авторів HTTP, визначив у своїй докторській дисертації набір архітектурних принципів для Web, котрі називаються REST. Ці принципи фокусуються на системних ресурсах, визначаючи як ресурси можуть бути адресовані та передані по HTTP.

Головною ідеєю REST було скоріш використання добре розвиненого HTTP для передачі даних між пристроями, ніж використання протоколу, побудованого поверх рівня HTTP, для передачі повідомлень. Додаток, розроблений слідуючи принципам даного архітектурного підходу, буде використовувати HTTP для виконання викликів між пристроями не спираючись на складні об'єктні механізми на кшталт CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call), чи SOAP. Відповідно, REST-додатки використовують функції HTTP запиту для надсилання, зчитування та видалення даних, використовуючи повну функціональність HTTP CRUD (Create, Read, Update і Delete) операцій. До того ж REST може використовувати HTTPS, надаючи безпечне передавання даних.

На відміну від SOAP, REST принципи легкі для розуміння та застосування розробнику, котрий знайомий з використанням HTTP. CRUD операції разом з функціями HTTP REST та відповідними SQL операціями показані у таблиці 2.

Таблиця 2

Відповідність HTTP-функцій SQL-операціям

CRUD-операції	Ключові слова REST (HTTP)	Оператори SQL
CREATE — створити чи додати нові сутності	POST	INSERT
UPDATE — оновити чи редагувати існуючі дані	PUT	UPDATE
READ — зчитати, отримати дані	GET	SELECT
DELETE — видалити інсуючі дані	DELETE	DELETE

Порівняльний аналіз протоколів

З огляду на те, що SOAP є похідним від XML-RPC і, водночас, виступає основною альтернативою REST, доцільним є порівняння технології SOAP окремо з XML-RPC та REST.

Порівняння SOAP та XML-RPC

Порівняно з SOAP, XML-RPC має простішу архітектуру, простіший для розуміння, ніж SOAP. Порівняння за основними характеристиками наведені у таблиці 3.

Таблиця 3

Порівняння SOAP та XML-RPC

Характеристика	SOAP	XML-RPC
Скалярні типи даних	Є	Є
Структури	Є	Є
Масиви	Є	Є
Іменовані масиви та структури	Є	Відсутні
Детальний опис помилок	Є	Є
Кодування, визначені розробником	Є (US-ASCII, UTF-8, UTF-16)	Відсутні
Типи даних, визначені розробником	Є	Відсутні
Можливість вказати отримувача	Є	Відсутня
Деталізація помилок	Є	Є
Потребує «розуміння» клієнтом	Так	Ні
Легкість практичного застосування та засвоєння	Потребує час для вивчення та розуміння	Легкий для розуміння і практичного використання

Порівняння SOAP та REST

У таблиці 4 наведено відповідності основних властивостей REST та SOAP.

Таблиця 4

Порівняння REST та SOAP

REST	SOAP
Передбачає модель обміну даними типу точка-точка, що виключає можливість використання у розподіленому середовищі з вузлами-посередниками	Розроблений для роботи у розподілених середовищах
Потребує мінімального набору інструментів, необхідна лише підтримка протоколу HTTP.	Потребує широкого набору інструментів та проміжного шару програмного забезпечення
URL ідентифікує ресурс, який отримується/видаляється/оновлюється	Вміст повідомлення визначає операцію
Формальні стандарти опису не набули широкого використання	Добре визначені механізми опису інтерфейсів, наприклад WSDL+XSD
Відсутність обмежень щодо корисного змісту	Корисний зміст має відповідати SOAP-схемі
Вбудована обробка помилок	Відсутність обробки помилок
Працює тільки з HTTP	Може використовувати різні транспортні протоколи
Менш детальний	Високий рівень деталізації

Надодачу, у архітектурному стилі REST застосовуються лише добре зарекомендовані стандарти, наприклад HTTP, SSL. Методи DELETE та PUT часто вимкнені фаєрволами, що призводить до складнощів з безпекою. Водночас SOAP підтримує широкий спектр стандартів для організації безпеки, надійності, підтримки транзакцій.

Використання протоколів

Так RPC та XML-RPC роками успішно демонструють доречність простоти своєї концепції в додатках, де акцент зроблено на швидкість передачі простих структур та виконання простих логічних операцій. В цьому випадку простота та обмеження виступають перевагами, оскільки вони значно зменшують труднощі реалізації протоколів і тестування їх сумісності. До того ж низький поріг складності вивчення дозволяє створювати функціональні додатки навіть початківцю. Надодачу, спираючись на стрімкий ріст популярності функціонального стилю програмування, можна зробити примущення, що стиль запитів у вигляді операцій, котрий надає RPC та XML-RPC, дає можливість описаним протоколам повернути популярність, властиву їм раніше чи набути ще більшого розповсюдження, ніж дотепер.

При виникненні потреби обміну складними структурами даних, формалізувати взаємодію клієнта та сервера, ввівши чіткі її контракти, доцільним є розглянути SOAP. Даний протокол слідує формамльним корпоративним підходам, добре сумісний з популярними великою кількістю транспортних протоколів, підтримка безпеки та авторизації є вбудованою функцією протокола, присутня можливість повного опису з використанням WSDL, що дозволяє реалізувати взаємодію типу точка-точка у мережі і надає можливість іншим пристроям у мережі «розуміти» веб-сервіс без втручання людини. Але даний протокол не позбавлений недоліків: не є доречним його впровадження у системах з повільним з'єднанням; протокол має труднощі реалізації, і, як наслідок, віне непопулярний серед веб-розробників та розробників мобільних додатків. Тобто не варто використовувати SOAP, коли треба надати API стороннім розробникам або коли пропускна здатність Інтернет-з'єднання низька. SOAP широко використовується у фінансових сервісах, платіжних системах та комунікаційних сервісах. Повсюдне використання даного протоколу у цих сферах дає впевненість у подальшій його популяризації та використанні у майбутньому.

Альтернативою протоколу SOAP є архітектурний стиль REST, що слідує філософії «Відкритого Інтернету». Його перевагами є відносна легкість у реалізації, чітке розділення серверної та клієнтської реалізацій, можливість кешування відповідей та різні формати відповідей на запити, як-то JSON чи XML. Так як REST напряму залежить від HTTP, для передачі даних між клієнтом та сервером треба менше зусиль, порівняно з SOAP чи RPC–XML. Переваги REST зробили його вибором багатьох розробників соціальних мереж, медіа-сервісів, мобільних додатків. Корпорації накшталт Twitter, LinkedIn та Slack використовують REST, як основу своїх додатків. Недоліками REST є те, що він працює лише поверх HTTP та складність впровадження безпеки та авторизації поверх нього. Цей підхід не варто використовувати, коли вимагається наявність чіткого контракту між клієнтом та сервером або відбувається виконання транзакцій, котрі включабть багаторазові виклики.

Висновки

Вибір протоколів взаємодії є важливим архітектурним рішенням і потребує значної уваги. Результат цього вибору має великий вплив на межі подальшого розвитку проекту та висуває ряд вимог до архітектури додатку. Безперечно, кожна з технологій має переваги та недоліки, але аналіз привів до висновку, що розглянуті протоколи є різними підходами і всі вони можуть бути доречними та принести користь, розмір якої залежить в першу чергу від контексту.

Сфери використання XML-RPC, SOAP та REST варіюються від невеликих систем з простою логікою до маштабних корпоративних додатків. Часто використовується комбінування даних технологій для досягнення оптимальної продуктивності чи надання різних API для задоволення потреб як розробників, котрі його використовують, так і інших систем, що взаємодіють з веб-сервісом.

На основі аналізу визначено у яких випадках доречніше використовувати ту чи іншу технологію, описані переваги та недоліки кожного з підходів. З огляду на це, можна сказати, що вибір пасуючої технології залежить в першу чергу від цілей та задач, котрі буде реалізувати система, долгостроковості її роботи.

Підсумовуючи проведений аналіз, можна сказати, що не існує наріжного каменя у виборі протоколу взаємодії. Для вибору максимально пасуючого протоколу варто аналізувати задачі, котрі буде вирішувати система, визначити перспективи розвитку розроблюваного проекту.