УДК 004.728.8

К.І. Яценко

ОРГАНІЗАЦІЯ ПРОТОКОЛІВ ПРИКЛАДНОГО РІВНЯ ДЛЯ ВИКОРИСТАННЯ МОБІЛЬНИХ ПРИСТРОЇВ В ГЕТЕРОГЕННОМУ СЕРЕДОВИЩІ

Запропоновано підхід до створення нового класу протоколів XDEP (XML Data Exchange Protocol) міжплатформної комунікації, мета якого полягає в стандартизації формату та методів обміну даними між серверними програмами та програмами, що працюють на переносних пристроях. Завдяки XML формату XDEP може бути легко використаний для комунікації між пристроями з різною архітектурою та має властивість гнучкого стандарту. В роботі представлено огляд найбільш популярних, з існуючих методів обміну даними та наводяться практичні приклади застосування нового класу протоколів.

Вступ

Мета побудови гетерогенних систем − визначення способу комунікації програмних ресурсів різних за логічною та фізичною структурою функціонування та їх інтеграція у єдину функціональну логічну систему. Основні проблеми при побудові таких систем випливають з визначення методів комунікації між різними програмними та апаратними середовищами. Найбільш широковживаним рішення цієї задачі є використання сокетів. Хоча таке рішення є універсальним підходом, воно недостатньо ефективне у кожному конкретному випадку. Далі наводяться причини, з яких видно що не вистачає методу використання сокетів та іншим існуючим підходам для повного вирішення проблеми комунікації різнорідних систем. У роботі розглядаються існуючі рішення, аналізуються їх переваги та недоліки і наводяться особливості побудованої системи з її практичною реалізацією.

Стандартний шлюзовий інтерфейс Common Gateway Interface (CGI) - це “протокол, що визначає процес взаємодії HTTP-сервера з програмами маршрутизації на сервері” [1]. Тобто, протокол CGI − це шлюз між HTTP-сервером та тими програмними засобами, які не можуть безпосередньо викликатися за протоколом HTTP. CGI-програми запускаються веб-сервером, який проводить хостінг (hosting) системи, тобто передає запити, отримані від користувача, CGI-програмам, які ці запити обробляють та повертають результати обробки користувачеві через веб-сервер. CGI-програма може бути інтерпретованою (написаною скрипковою мовою, здебільшого використовуються мови Perl та PHP) або відкомпільованою програмою (написаною будь-якою мовою програмування, як правило імперативною, наприклад C/C++, Visual Basic, Pascal, Fortran тощо). У цій технології є декілька ключових моментів. По-перше, вона не вимагає спеціальних програмних засобів на клієнтській частині (комп’ютері користувача), оскільки CGI-програми виконуються повністю на сервері та не залежать від клієнтського комп’ютера і одна від одної, з чого випливає друге − відсутність засобів контролю за роботою CGI-програм, що використовують спільні ресурси. Таким чином цілісну систему в рамках цієї технології побудувати важко [1].

Іншою технологією доступу до даних є прикладний програмний інтерфейс API (Application Program Interface), що являє собою “набір правил, які визначають процедури виклику функцій за допомогою програмного пакета” [2]. Це стандартна технологія роботи комп’ютерних програм, проте як Інтернет-технологія вона почала використовуватись не так давно. На прикладі моделі компонентних об’єктів COM (Component Object Model), програмний пакет API складається з однієї або декількох динамічних бібліотек DLL (Dynamic Linked Library) та компонента COM для доступу до них. На

відміну від технології CGI, API передбачає інтеграцію програмного пакета з усією системою. Ця технологія набагато складніша, ніж CGI, але водночас вона є й набагато потужнішою. Якщо технологія CGI дає змогу створити шлюз, за допомогою якого на сервері зможуть виконуватися ті чи інші програмні процеси, то за допомогою API сам клієнт може віддалено працювати з програмами на сервері і викликати потрібні функції не опосередковано через шлюз, а напряму. Тут важливими є декілька моментів. По-перше, ця технологія передбачає наявність на комп'ютері користувача спеціального клієнта для доступу до програм на сервері. По-друге, на відміну від CGI, існує контроль роботи всіх програмних компонентів API. Прикладні програми, бібліотеки, вебсторінки тощо складають єдиний комплект (assembly) взаємопов'язаних компонентів, що використовують єдине програмне ядро. До того ж, такий комплект інтегрується з операційною системою і використовує деякі її програмні компоненти, що значно покращує ефективність таких програм [3].

Можна вважати, що технологія CGI передбачає створення на сервері великої кількості маленьких програм, які працюють відносно незалежно одна від одної, а технологія API, навпаки, передбачає створення однієї цілісної системи, різні компоненти якої працюють взаємопов'язано. В порівнянні з технологією API, CGI-програми працюють набагато повільніше і вимагають більше серверних ресурсів. З іншого боку, CGI-програми невимогливі до клієнтського програмного забезпечення, водночас як API-технологія вимагає наявності на комп'ютері користувача спеціального клієнта API [4].

Розробникам програмного забезпечення, при використанні CGI, достатньо створити веб-публікацію, в якій доступ клієнта до даних здійснюється за допомогою форм, що обробляються CGI-програмами. Але, як вищеописано, потужну систему таким чином побудувати дуже важко через проблеми із синхронізацією даних, захистом інформації, безпекою сервера тощо. API-технологія більш прогресивна і вирішує наведені проблеми CGI.

З появою технології розподілених обчислень мережева взаємодія перейшла на новий щабель розвитку, коли різні установи отримали змогу координувати свої зусилля і більш раціонально розпоряджатися своїми ресурсами, використовуючи можливості одне одного. Так з'явилася технологія вебсервісів.

Веб-сервіси − це вид Інтернет-програм, що забезпечують динамічний зв'язок різнорідних систем в основі якого лежить використання спільних стандартів [5]. Вебсервіс являє собою програмний компонент або інформаційний ресурс, доступ до якого в мережі Інтернет/Інтранет/Екстранет здійснюється через стандартні веб-протоколи. Важливим тут є те, що веб-сервіси дозволяють отримувати доступ до програмних компонентів незалежно від того, як ці компоненти працюють, на якій платформі та які пристрої використовують, тому що веб-сервіси використовують для обміну даними протокол HTTP та формат XML, які статли загальноприйнятими. В результаті поєднання технологій HTTP та XML був отриманий протокол SOAP (Simple Object Access Protocol), який і став основою веб-сервісів. Отже, на появу технології веб-сервісів вплинули дві тенденції сучасної програмної індустрії: прийняття HTTP як стандартного протоколу, за допомогою якого здійснюється доступ в Інтернет, та визнання формату XML фактичним стандартом для передачі даних.

Веб-сервіси базуються на трьох основних веб-стандартах:

- *протокол SOAP* - стандарт для обміну (відсилання та отримання) повідомленнями між сервісами. SOAP являє собою XML-конверт із запитом (або відповіддю на запит), у якому зазначені засоби представлення XML-структури та засоби зв'язку за протоколом HTTP [5];
- *мова WSDL* (Web Services Description Language) - засіб опису програмних інтер-

фейсів веб-сервісів. WSDL-опис являє собою документ формату XML, який містить дані про протокол взаємодії, адреса сервісу, формат конверта SOAP та його елементи [6].

- *стандарт UDDI* (Universal Description, Discovery and Integration) - це, по суті, реєстр веб-сервісів. За допомогою цього стандарту певний веб-сервіс може бути зареєстрований у веб-реєстрі UDDI для його подальшої ідентифікації серед багатьох інших. Зацікавлена організація може обрати з цього реєстру ті сервіси, якими вона хоче користуватися, та зареєструвати створені нею сервіси, щоб ними могли користуватись інші зацікавлені організації [7].

Веб-сервіси − це засіб інтеграції, яка має декілька рівнів. Перший рівень – рівень даних на якому програми обмінюються інформацією. Наприклад, програма розташована на одному сервері, може використовувати дані з бази даних на іншому. Цей рівень передбачає інтеграцію даних, що є найпростішим видом інтеграції. Другий рівень – рівень об'єктної взаємодії. Тут мова йде про те, що програма, розташована на одному сервері, може запускати програмні процеси на іншому. Найскладнішим є третій рівень інтеграції, або інтеграція на рівні стандартної семантики. На цьому рівні сервіси "спілкуються" спільною мовою [7], обходячи технологічні розбіжності. Один сервіс може звертатись до іншого, наприклад, із запитом на виконання пошуку, запитом на отримання статистики та ін. На цьому рівні інтеграції сервіси потребують лише стандартизації семантики. В разі, якщо семантичних розбіжностей між ними немає, інтеграція розрізнених систем є дуже простою. Така стандартизація семантики досягається із використанням специфікації WSDL, яка описується системно-незалежною мовою. З одного боку, системна незалежність програм забезпечується використання мови XML при створенні WSDL-описів, а з іншого − специфікацією SOAP, що дозволяє взаємодіяти серверній та клієнтській програмам. Клієнта серверні- програми лише надають дані, а протокол SOAP повністю реалізує обмін цими даними [6]. Технологія веб-сервісів надає суттєву гнучкість при конструюванні розподілених програмних комплексів, вона має деякі недоліки. Основні з них: неоднозначність специфікації SOAP, недостатній рівень безпеки (відсутність шифрування даних на рівні протоколу) та невисока швидкість роботи.

Особливо увагу викликають механізми безпеки протоколу SOAP. Коли мова йде про невеликий проект, аналізаторів безпеки, передбачених протоколом SOAP, може цілком вистачити. Але при розробці масштабного Інтернет-проекту, з розгалуженою системою об'єктів, схем та сценаріїв доступу, розробники можуть зіткнутися з проблемою недостатності стандартних процедур безпеки. В цьому випадку їм доведеться самостійно програмувати велику кількість додаткових засобів: аналізаторів датаграм, фільтрів та ін. Інший недолік − низька швидкість роботи веб-сервісів заснованих на SOAP, що є вже більш складною проблемою, хоча вона притаманна всій технології розподілених обчислень, а не лише веб-сервісам. У даному випадку неможливо розробити власні методи для зменшення обсягу технічної інформації, тому що будь-які модифікації вплинуть на стандарт протоколу. Вирішення цієї проблеми безпосередньо залежить від потужності серверів та пропускної здатності каналів зв'язку. В даному випадку до часу з'єднання клієнта із сервером додається час з'єднання одного сервера з іншим. До того ж, менш потужний із серверів стає "вузькою ланкою" взаємодії. На останку − неоднозначність специфікації SOAP [8]. Різні інтерпретації SOAP по-різному, іноді суперечливо, трактують деякі елементи. Специфікація, яка розроблена для стандартизації веб-орієнтовного програмування, не завжди забезпечує сумісність різних систем на практиці. Маємо ситуацію, коли є стандарт, але проблема стандартизації ще не вирішена. Веб-сервіси, розроблені різними організаціями, на практиці цілком можуть виявитись несумісними один з одним.

De-facto стандартними засобами розробки веб-сервісів стали дві великі платформи: .NET Framework від компанії Microsoft та Java 2 Platform, Enterprise Edition (J2EE), основним розробником якої є компанія Sun Microsystems Inc [9]. Обидві платфирми надають схожі методи роботи. Найголовною їх вимогою є наявність специфічних бібліотек на стороні клієнтських програм для використання веб-сервісів. Інтеграція портативних пристроїв, у даному разі, викликає складності, через відсутність таких програмних бібліотек для багатьох з них. Наприклад, можливо побудувати програму, яка використовує веб-сервіси з операційної системи Microsoft Pocket PC® або MS Windows Mobile ® з використанням MS .NET Compact Framework. Але в разі, якщо підприємство хоче використати пристрої власного виробництва, то реалізація стандарту веб-сервісів стає складною з технічної та програмної точок зору. На сам перед – це необхідність розробки великої кількості програмних бібліотек для підтримання протоколів TCP/IP та SOAP.

Протокол XDEP

XDEP (XML Data Exchange Protocol) – це клас протоколів прикладного рівня для обміну даними, який підтримує аунтентифікацію та авторизацію. Особливість цього класу є підтримка сесії користувача без необхідності підтримання постійного з'єднання в режимі реального часу. Клас протоколів описує логічні вимоги до методів та структур даних під час взаємодії програм в гетерогенному середовищі. Задачі, які вирішується при використання цього протоколу:

- мулітиплатформість клієнт-програм;
- захищеність даних від несанкціонованого доступу через авторизацію та аунтентифікацію.

Протокол складається з запитів двох рівнів доступу: глобального та сесійного. Кожен рівень визначає тип даних доступних користувачам. Наприклад, технічна інформація про сервер може бути отриманна без аунтефікації програми клієнта. Така інформація належить до глобального рівня доступу. Водночас, дані з бази даних можуть бути представлені лише зареєстрованим користувачам. Клас протоколів передбачає обмін запитами (та відповідями на них) між клієнтами та сервером. На кожний отриманий запит сервер генерую відповідь відповідного рівня доступу. Не залежно від результату обробки запиту, сервер має повідомити клієнта про те, що запит було отримано та опрацьовано. Як для запитів так і відповідей пропонується наступна схема спеціалізованого формату.

Глобальний рівень команд

Команди цього рівня доступу надають доступ до відкритих даних, інформації доступною будь-якій програмі, якщо вона може коректно сформувати запит.

Сигнатура запитів

Загальний формат запитів глобального рівня:

<[CommandName] Version="string" RequestID="string" devid="device id">
<[RequestSpecificParams]>
</[RequestSpecificParams]>
</[CommandName] >

[CommandName] – назва команди, яка має бути унікальною у межах логічної системи. Кожен протокол підтримує версіонізацію команд, що дозволяє обробляти команди з однаковим ім'ям, але різною структурою, тобто поліморфні команди. Для ідентифікації запиту в межах сервера разом з ім'ям використовується параметр версії команди. Необхідність поліморфності команд полягає у можливості розвинення систем з одночасною підтримкою старих версій протоколів (обслуговуванням існуючих клієнт-програм).

[RequestSpecificParams] − XML структура даних команди. Виходячи з специфікації XML, обмеженість на ієрархію даних відсутня, отже команда не має обмежень на обсяг інформації, що передається. Атрибути команди наведено в табл. 1.

Таблиця 1. Атрибути команди

Назва атрибута	**Опис**	**Значення по замовчуванню**	**Альтернативний**
Version	Версія реалізації запиту	Поточна версія команди	Так
RequestID	Унікальний номер запиту в межах клієнт-програми	0	Ні
DeviceID	Унікальний ідентифікатор клієнт-програми в межах сервера	Відсутня	Ні

Version − використовується для визначення версії команди в системі сервера;

RequestID – унікальний номер запиту в системі клієнта. Це альтернативний атрибут і не впливає на виконання команди в межах серверу. Але, його наявність необхідна в разі, якщо клієнт – програма працює у режимі паралельних запитів, з метою однозначної ідентифікації запитів та відповідей.

Сигнатура відповіді

<[CommandName]Resp RequestID="string“ Code="000" Msg="OK”>
<[RespSpecificParams]>
</[RespSpecificParams]>
<[System]>
<[SystemSpecificParams]>
</[SystemSpecificParams]>
<[/System]>
</[CommandName]Resp >

Аналогічно до структури команди, відповідь представлена в XML форматі.

[CommandName]Resp – ім’я відповіді складається з ім’я запиту та слова *Resp* (це рекомендація, яка визначає добру практику);

[RespSpecificParams] – тіло відповіді команди;

[System] − передача технічної інформації клієнту. Наприклад, у разі зміни IP адреси комутуючого сервера клієнт-програми має бути моментально проінформований. Сервер не може самостійно встановити зв’язок з клієнт-програмами, тому цей спеціальний тег було добавлено в сигнатуру відповіді. В табл. 2 наведено атрибути відповіді.

На відміну від запитів, відповіді розширені додатковими атрибутами. Змістом цих атрибутів є представлення результатів виконання запиту.

RequestID – ідентифікатор команди;

Code – код помилки. Далі наведено стандартні коди помилок;

MSG – опис помилки. Надає контекстний опис для представлення користувачеві.

Сесійний рівень команд

Повідомлення цього рівня використовуються для надання закритої інформації. Можливість роботи сесії користувача в режимі без постійного з’єднання глобальні запити забезпечується додатковими атрибутами ідентифікатора сесії. Для отримання цього ідентифікатору введена спеціальна команда глобального доступу *OpenSession*. Далі, наведено цю команду:

<OpenSession Version=”string” RequestID="string""">
<devid></ devid>
<username></username>
<password></password> [encoded]
</OpenSession >

Таблиця 2. Атрибути відповіді

Назва атрибута	**Опис**	**Значення по замовчуванню**	**Альтернативний**
RequestID	Унікальний номер запиту в межах системи клієнта	0	Ні
Code	Код помилки	0	Ні
MSG	Опис помилки	Порожній рядок	Так

Вузол *devid* зберігає унікальний ідентифікатор приладу. Додаткова перевірка потрібна для того, щоб користувач міг отримати доступ до закритих серверних даних лише через прилади (комп'ютери) зареєстровані на сервері. Ця додаткова перевірка додає ще один рівень захищеності даним, у разі, якщо користувач втратив свій обліковий запис та третя особа намагається його використати на іншому приладі.

Відповідь на цей запит:

```
<OpenSessionResp RequestID="string"
Code="000" Msg="OK">
        <SessionGUID>123kjhsad908h@#q389qgm
        </SessionGUID>
</OpenSessionResp >
```

У результаті запиту *OpenSession* клієнту надається унікальний ідентифікатор сесії у межах серверу. Методи генерування ідентифікатору сесії не мають значення. Завдяки цьому ідентифікатору клієнт-програма отримує доступ до закритої інформації.

Сигнатура запиту

```
<[CommandName] Version="1" RequestID="31“
SessionGuid =“string” devid=”device id”>
        <[RequestSpecificParams]>
        </[RequestSpecificParams]>
</[CommandName] >
```

Формат відповіді на запити сесійного рівня співпадають з форматом відповіді глобального рівня.

Обробка помилок

XDEP стандартизує обробку помилок, та надає наступний перелік стандартних помилок. Коди від 0000 – 0999 є зарезервованими кодами класу. В табл. 4 наведено коди найбільш характерних помилок.

Криптування даних

На відміну від SOAP протоколу, який не передбачає спеціальних тегів для виділення закодованої інформації, XDEP

Таблиця 3. Атрибути команди

Назва атрибута	**Опис**	**Значення по замовчуванню**	**Альтернативний**
Version	Версія реалізації запиту	Поточна версія команди	Так
RequestID	Унікальний номер запиту в межах клієнт-системи	0	Ні
SessionGuid	Унікальний ідентифікатор сесії	Відсутнє	Ні
DeviceID	Унікальний ідентифікатор клієнт-програми	Відсутня	Ні

Таблиця 4. Зарезервовані коди помилок

Код помилки	**Опис**
0000	Помилка відсутня
0001	Невідомий запит
0002	Помилка формату атрибутів
0003	Помилка параметрів атрибутів
0004	Помилковий формат полів
0005	Відсутні закриваючі теги параметрів
0800	Неможливо відкрити сесію
0801	Користувач або пароль невірні
0802	Невідомий ідентифікатор сесії
0998	Серверна помилка
0999	Невідома помилка

надає такий інструментарій. А саме, вводяться теги *<Encrypted>,* що «обгортають» запити та відповіді. Методи кодування даних визначаються в рамках кожної реалізації протоколу.

```
<Encrypted>
        < Content >
        </Content >
</Encrypted>
```

Практика використання

На поточний момент наведений клас було вже декілька разів успішно використано в проектах різних за своєю суттю, що додатково продемонструвало потужність та зручність використання самостійно побудованих протоколів за наведеними стандартами. Розглянемо більш детально один з таких прикладів.

У рамках проекту Personal Tracker (http://gps.uktbgroup.com), що є підсистемою великої навігаційної системи Sealand EyeSat® (www.sealand.it), вирішено побудувати протокол «спілкування» центрального сервера та віддалених пристроїв з вбудованими GPS навігаторами за стандартами XDEP. Зміст проекту полягає в наданні користувачам можливості віддаленого спостереження за мобільними об'єктами: автотранспортом та людьми. Наведемо приклад основної команди протоколу, яка несе в собі географічні характеристики об'єкта та є головним запитом системи.

```
<twreq stamp="rmc" devid="device id"
[queryresult=1] >
        <lat d="N | S">latitude</lat>
        <long d="E | W">longtitude</long>
        <sats>A | V</sats>
        <satt>satellite timestamp</satt>
        <clt>client side timestamp</clt>
        <spd>horizontal speed </spd>
        <dirc>true direction of mobile client </dirc>
        [<alt>altitude</alt>]
</twreq>
```

У цілях економії трафіку, сервер не відправляє відповідь про отримання команди, але в разі, якщо це необхідно клієнту, то він інформує сервер додаючи атрибут *queryresult = 1* (клас протоколів XDEP не забороняє введення нових атрибутів). У разі, якщо значення атрибуту менше за одиницю, то сервер ігнорує його.

Повний приклад одного з запитів:

```
<twreq stamp="rmc" devid=
="FCCB64B3-1F94-4a37-992F-620A9E8B64FB">
        <lat d="N">50.27.00</lat>
        <long d="E">30.31.24</long>
        <sats>A</sats>
        <satt>06.12.2006;01:03:49</satt>
        <clt>06.12.2006;03:02:30 </clt>
        <spd>0.0</spd>
        <dirc>0.0</dirc>
</twreq>
```

Цей запит інтерпретується таким чином: пристрій із ідентифікатором FCCB64B3-1F94-4a37-992F-620A9E8B64FB знаходиться в 50° 27 хв. 00 сек. північної широти та 30° 31 хв. 24 сек. східної довготи (центр міста Києва, Україна). Час супутника – 1 год. 3 хв. та 49 сек., 6 грудня 2006 року, час пристрою – 3 год. 2 хв. та 30 сек., 6 грудня 2006 року. Прилад не рухається, азимут становить 0° (не визначено). У даному прикладі відсутній атрибут *queryresult* тому сервер не поверне повідомлення про прийняття та обробку запиту. З метою підвищення безпеки, та для попередження атак на сервер, система не повертає коди помилок у разі, якщо прилад не був зареєстрований в системі, або у разі, якщо обліковий запис користувача та його пароль невірні.

Під час проектування системи розглядались різні методи впровадження комунікації між віддаленими системами. Ці методи порівнювались за наступними характеристиками: методи представлення даних (бінарний чи текстовий формати); методи встановлення комунікації (рівень транспортних протоколів); можливості оновлення протоколів без зупинення роботи

систем (оновлення команд, розширення набору команд); простота імплементації системи на базі обраного протоколу; кількість платформ, що підтримується та можливість розширення переліку клієнт-пристроїв пристроями власної розробки. Найбільш придатними для реалізації цієї функціональності розглядались такі методи комунікації: Microsoft .NET Remoting, Web Services, Сокети із використанням XDEP.

Microsoft .NET Remoting – технологія яка працює лише на системах класу Windows®, тому її використання не дозволяє впровадження власних пристроїв або використання існуючих пристроїв на базі платформ відмінних від Windows®.

Веб-сервіси – на перший погляд вдале рішення, але для конкретної задачі бракує версіонізації запитів, та присутня надлишкова кількість технічної інформації у рамках протоколу SOAP, що збільшує кількість трафіку під часу обміну даними, і вимагає наявності стандартних бібліотек, або складну власну розробку комунікації на базі SOAP. Важливим було зауваження неоднозначності специфікації SOAP протоколу. В зв'язку з тим, що кожний виробник програмних бібліотек під різні платформи реалізовує специфікацію згідно власної інтерпретації, був великий ризик несумісності роботи приладів на платформі Java та MS .NET Framework.

Сокети із використанням XDEP – найбільш придатне вирішення поставленої проблеми. Всі існуючі пристрої з методами мережевими можливостями, в рамках конкретного проекту, підтримують IP протокол (на базі GSM мережі GPRS- протокол). Окрім того, XDEP – самодостатній стандарт який не вимагає додаткових бібліотек. Ще однією перевагою використання протоколу є наявність стандартних методів конвертації класів програм в XML формат тим самим суттєво спрощуючи модель розробки. Щодо неоднозначності стандартів, то XDEP стандартизує лише зовнішню структуру даних описуючи тим самим метадані, при цьому жодним чином не впливаючи на представлення даних у середині команд.

Висновки

XDEP представляє клас протоколів обміну даними. Під час практичного прикладу було продемонстровано переваги XDEP над іншими популярними технологіями. Визначений як клас протоколів XDEP надає стандарти розробки власних протоколів під час реалізації програм в гетерогенному середовищі. На відміну від універсальності веб-сервісів або обміну бінарної інформації через сокети, клас протоколів представляє набір рекомендацій, які можуть допомагати в більш ефективній реалізації власних методів комунікації програм, які інтегруються в єдину систему, але працюють на різних платформах. Ця особливість суттєво впливає на зручність використання протоколів побудованих за принципами XDEP.

Перше комерційне застосування, з використанням протоколу обміну даними побудованого на базі XDEP, було реалізовано в середовищі мобільних пристроїв. Але це не обмежує використання методології розробки власних протоколів лише на мобільну платформу, хоча XDEP розроблявся саме в рамках задачі інтеграції мобільних пристроїв.

Аналізуючи конкретні реалізації можна виділити наступні переваги: простота програм з використанням власного протоколу обміну даними, підтримка сесій роботи в неоперативному режимі (off-line), формат запитів, зручний для перевірки та відлагодження програм. До недоліків відносяться: значна кількість технічної інформації у запиті (становить до 60% – недолік для текстового формату XML від загального обсягу повідомлення); необхідність адаптації протоколу під час нової реалізації.

1. *Shishir Gundavaram*. CGI Programming on the World Wide Web (Nutshell Handbook), 1996
2. *John E. Swanke*. COM Programming by Example: Using MFC, ActiveX, ATL, ADO, and COM+ (with CD-ROM), 2000.
3. *Dale Rogerson*. Inside Com (Microsoft Programming Series), 1997.
4. *Andrew Troelsen*. COM and .NET Interoperability, 2002.
5. *Will Iverson*. Real World Web Services, 2004.
6. *Eric Newcomer*. Understanding Web Services: XML, WSDL, SOAP, and UDDI, 2002
7. *Steve Graham, Doug Davis, Simeon Simeonov, and Glen Daniels.* Building Web Services with Java : Making Sense of XML, SOAP, WSDL, and UDDI (2nd Edition) (Developer's Library), 2004.
8. *Damien Foggon, Daniel Maharry, Chris Ullman, and Karli* Watson. Programming Microsoft .NET XML Web Services (Pro-Developer), 2003.
9. *Mohamed E. Fayad, Ralph E. Johnson, and Douglas C. Schmidt.* Building Application Frameworks: Object-Oriented Foundations of Framework Design, 1999.

Отримано 15.01.2007

***Про автора*:**
Яценко Кирило Ігоревич,
аспірант факультету кібернетики Київського національного університету
імені Тараса Шевченка,
e-mail: author@kirillyatsenko.com