УДК 004.75

ТЕЛЕНИК С.Ф.,
ТРОЦЬКИЙ С.О.

ВИКОРИСТАННЯ КОМПОНЕНТНОГО ПІДХОДУ ДО ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНИХ АДАПТИВНИХ СЕРВІСІВ

В останні роки розробка на основі компонентів стала визнаним підходом. Розглянуто метод, що дозволяє спроектувати сервіс, вихід з ладу якого, призводить до автоматичного перерозподілу навантаження в системі. Компонентом IAC-сервісу є бібліотека (файл з розширенням dll), яка написана для .NET Framework та має головний клас, що реалізує деякий API. Компоненти зв'язуються на етапі виконання між собою за допомогою декларацій в конфігураційному файлі. Завантажувач компонент автоматично ініціює компоненти згідно конфігурації.

In recent years component-based development has become an established approach. The method, which allows to design services in a such way, that failure of any leads to an automatic load redistribution is considered in the article. Components of IAS service are the libraries (files with extension dll), written for the .NET Framework, and which have the main class that implements specific API. The components are linked to each other at runtime based on configuration file. Loader of the components automatically initiates the components according to the configuration.

Ключові слова: компоненто-базований підхід, сервіс-орієнтована архітектура, модуль, проектування

Вступ

В умовах розвитку сервіс-орієнтованого підходу фінансові установи переходять до єдиної методології управління ризиками. При цьому розрахунки ризиків, потенційних втрат та інших важливих для прийняття рішень показників у сучасних фінансових установах базуються на документі Базель III. Цей документ Базельського комітету з банківської нагляду містить відповідні методичні рекомендації в сфері банківського регулювання. Усталений підхід до проектування інформаційних систем (ІС) банків передбачає, що всі розрахунки з одного напряму діяльності (підсистеми, функції, бізнес-процесу) здійснюються в одній системі у зоні відповідальності одного департаменту, а їх результати потім використовуються всіма іншими підсистемами, функціями, бізнес-процесами) банку у зоні відповідальності різних департаментів.

З точки зору побудови ІС банків на єдиній інфраструктурі інформаційних технологій (ІТ-інфраструктурі) при реалізації такого підходу виникає низка проблем. За складністю і терміновістю виділяються проблеми проектування (створення, вибору, модифікації) сервісів та інтеграції як такої системи з іншими системами, наприклад системою HFT (high frequency trading), так і її компонентів між собою. Ці складність і терміновість викликані високим рівнем вимог до ІС банків загалом і важливістю з огляду на потреби ефективної реалізації сервісного підходу в системах такого класу. Насамперед потрібно особливу увагу приділити забезпеченню високих вимог до таких параметрів системи:

1) Максимальний час відповіді з врахуванням часу передачі даних (при їх великих об'ємах);
2) Відмовостійкість, легкість підтримки і масштабованість;
3) Можливість розгортання системи в різних географічних зонах та ін.

Загальна процедура під час виходу з ладу банківської системи (її підсистеми) полягає у надсиланні до команди її підтримки відповідного повідомлення (за допомогою програми автоматичного фіксування виходу системи з ладу) і реакції на нього команди підтримки (в разі необхідності перехід на деякий час на запасну систему (підсистему) – DR (disaster recovery). Недоліком процедури є те, що вона не сприяє задоволенню високих вимог до параметрів ІС.

Постановка задачі

Інтелектуальні адаптивні сервіси (IAC) – це наділені інтелектуальними можливостями сервіси, здатні перерозподілити функціональність і

навантаження між вузлами (сервісами) IT-середовища (на сьогодні це хмари), в якому вони базуються. Саме сервіси цього типу можна розглядати як основу надання ІС банків нових якостей. З іншого боку, їх можна розглядати як архітектурний елемент компоненто-базованого підходу до проектування, реалізації і експлуатації ІС банків, основу нової гнучкої і ефективної методології створення, експлуатації і модернізації ІС. Крім того, це ще й динамічний сектор ринку IT. Ефективне виконання сервісів IAC забезпечують системи, що мають архітектуру, зображену на рисунку 1.

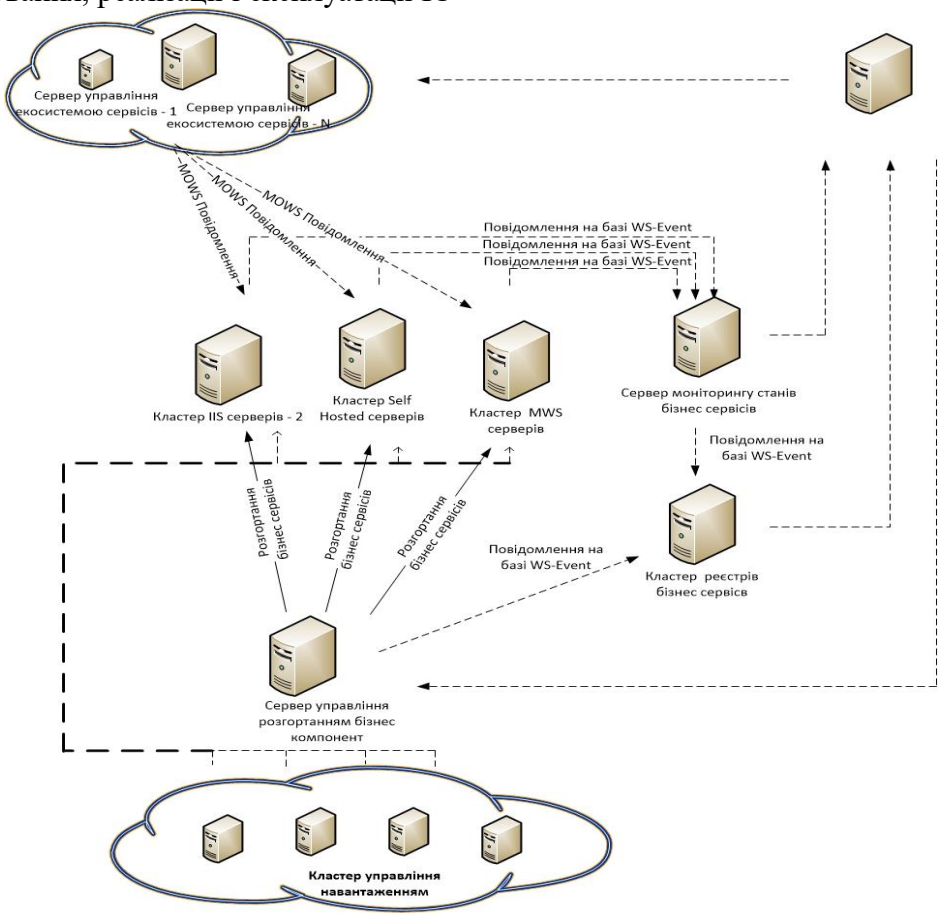

Рис. 1 Загальна архітектура системи виконання IAC сервісів

Важливими для забезпечення високого рівня вимог до ІС банків особливостями цих сервісів є автоматичне детектування збою сервісів, відповідальних за бізнес-процеси, перерозподіл бізнес-процесів між іншими сервісами, перезапуск згідно конфігурації модуля компонента, який відповідає за виконання бізнес-процесу, можливість виконання додатків, що були створенні для .NET Framework в режимі сервісу SOA згідно BPML документу, навіть якщо ці додатки не бути спочатку створені для виконання в режимі сервісу.

Якщо бізнес-процес має певний внутрішній стан і виникає проблема під час його виконання, модуль компонента, який відповідає за виконання бізнес-процесу, ініціює його перезапуск згідно конфігурації, навіть за використання ланцюга бізнес-процесів.

Компонентом сервісу IAC є написана для .NET Framework бібліотека (файл з розширенням dll), яка має головний клас, що реалізує деякий API. Компоненти зв'язуються між собою на етапі виконання за допомогою декларацій в конфігураційному файлі. Завантажувач компонент автоматично ініціює їх згідно конфігурації. Модуль компонента – це логічна програмна абстракція, шо виконує специфічну для компонента функцію. Кожен компонент має декілька (від 1 до *K*) модулів в залежності від конфігурації. Різні компоненти можуть використовувати той же модуль. Кластер управління навантаженням може містити декілька (від 1 до *N*) серверів, в залежності від максимальної дозволенної затримки згідно специфікації.

Огляд існуючих рішень

Проблема інтеграції розрізненої інформації сервісів з метою її подальшого опрацювання та прийняття рішень на її основі постала разом із

появою сховищ даних, ще у 80-х роках минулого століття. Передумовою її виникнення було зростання інтересу до розподілених баз даних та їх масове впровадження у бізнесові структури [1]. У цій же праці визначені методи роботи з джерелами з різними структурами даних:

1. Пошук інформаційних джерел на основі метаданих.
2. Відбір джерел з використанням онтологічних визначень.
3. Інтеграція на рівні сховищ даних.

Оскільки кожен IAC сервіс має свій набір метаданих, то найкращим вибором буде метод пошуку сервісів на основі метаданих. Оскільки для специфікації метаданих сервісу використовують різні типи метаданих, то тип метаданих, визначений для описання сервісу, є частиною його моделі вимог і може не збігатись з типами метаданих інших сервісів [1].

Формальні моделі, підходи і інструменти інтеграції програмних застосувань можна знайти у багатьох літературних джерелах, зокрема [2-4].

Але сьогодні постає потреба в технологіях компоненто-базованого проектування і експлуатації інформаційних систем в розвинених IT-середовищах на основі поєднання сервіс-орієнтованого, процесного і композитного підходів.

Загальний підхід до вирішення проблеми

Сервер управління розгортанням бізнес-компонентів формує сервіс, який складається з: WS-компонентів загального використання (які базуються на WS-стандартах); компонентів, що інкапсулюють бізнес-логіку процесу (побудованих за специфікацією BPEL4WS + BPML); компонентів, які реалізують програмні додатки та розширення специфікацій. Загалом сервіс складається з 16 компонентів, основне призначення яких пов'язане з реалізацією певної функціональності, тобто з виконанням програмних модулів підтримки бізнес-процесів. Але в умовах побудови сервісів на засадах SOA вони повинні мати низку компонентів, що відповідають за ефективне управління внутрішнім станом самого сервісу і його взаємодію з іншими сервісами. Сервіс складається з таких рівнів:

- Прикладний;
- Сервісний;
- Комунікаційний.

Загальна структура сервісу зображена на рисунку 2.

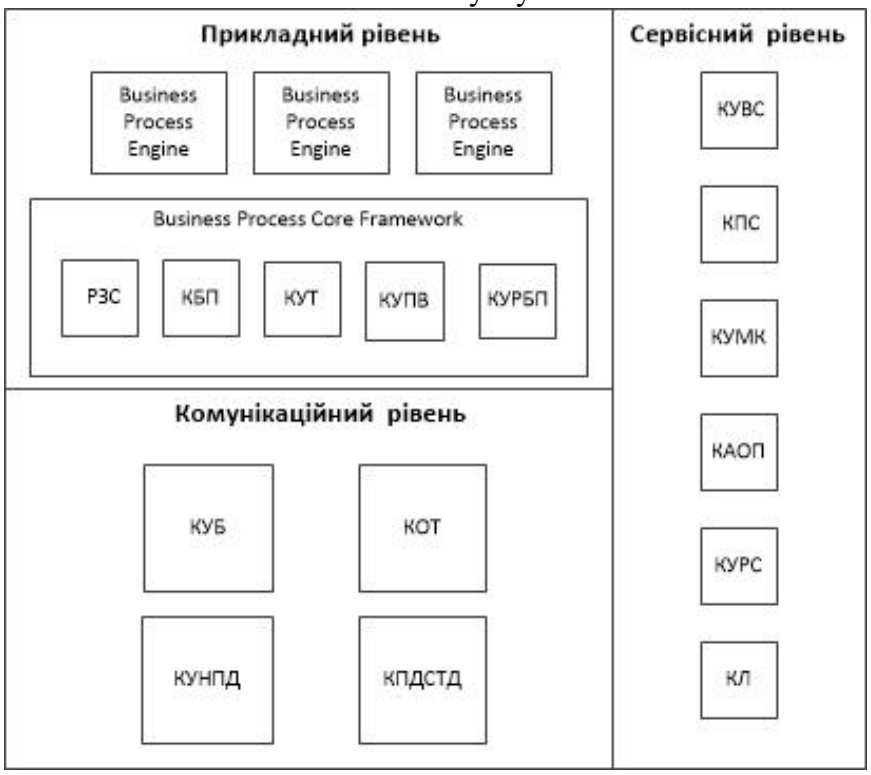

Рис. 2 Структура сервісу, розділеного на компоненти

Оскільки сервіс і його компонентна структура є найважливішими складовими компонентно-базованого підходу, розглянемо структура сервісу, розділеного на компоненти більш детально. Це дозволити не тільки описати призначення і можливості кожного компоненту, але й уточнити проблеми реалізації сервісів та шляхи їх вирішення, насамперед окреслити математи-

чні моделі й методи, здатні забезпечити заявлені можливості компонентів.

Розпочнемо з прикладного рівня, який відповідає за виконання програмних модулів бізнес-процесів, що описуються за допомогою BPEL4WS + BPML. Саме тут в програмних модулях реалізована певна функціональність, обумовлена потребами підтримки певних бізнес-процесів

Реєстр зв'язаних сервісів (РЗС) – компонент, який взаємодіє з КПС та на основі правил внутрішньої мови специфікації доменів – domain specific language (DSL), зв'язує себе та інші сервіси в кластері.

Кеш бізнес процесів (КБП) – компонент що реалізує буфер з внутрішнім станом бізнес-процесу на випадок перезавантаження процесу, його зупинки та старту. Завантажувач бізнес процесів виконується один раз під час запуску сервісу.

Компонент управління послідовностями викликів (КУПВ) – якщо бізнес-ланцюг складається з викликів від 1 до M бізнес-процесів, цей компонент зберігає інформацію про сервіс (бізнес-процес), до якого був зроблений виклик, щоб отримати дані, та координує повторні запити, якщо дані не надійшли.

Компонент управління транзакціями (КУТ) – компонент прикладного рівня, який застосовується, якщо бізнес процес або декілька бізнес процесів, що виконуються як єдине ціле потребують підтвердження атомічного виконання.

Компонент управління ресурсами бізнес-процесів (КУРБП) – компонент, який разом з КУРС забезпечує ефективне використання відповідних ресурсів.

Наступний розглянемо сервісний рівень, який відповідає за управління внутрішнім станом сервісу, забезпечує оптимізацію функціонування сервісу.

Компонент управління внутрішнім станом (КУВС) – компонент, який відповідає за збереження внутрішнього стану, його передачу в разі необхідності при виході з ладу до диспетчера бізнес-сервісів, аналізу та збору внутрішніх характеристик та метрик.

Компонент пошуку сервісів (КПС) – компонент, який відповідає за пошук і визначення можливостей взаємодії з новими сервісами для підтримки певної функціональності.

Компонент аналізу та оптимізації продуктивності (КАОП) – компонент, який у взаємодії з компонентом КУВС відповідає за підтримку часу реакції сервісу на заданому рівні згідно SLA (service level agreement).

Компонент управління міжсервісним контекстом (КУМК) – компонент, який застосовується у разі необхідності передачі бізнес-процесу чи його частини (активності) зі збереженням стану та дані між сервісами. Схема передачі контексту зображена на рисунку 3. Літерою М позначаються повідомлення (messages), літерою К – контексти, що передаються між сервісами.

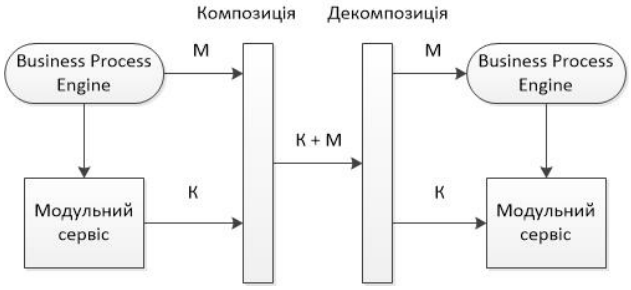

Рис. 3 Схема передачі контексту

Розширення функціональності будь-якого рівня зводиться до створення нових компонентів при одночасній декларації їх інтерфейсів. Завантажувач компонентів динамічно завантажує та ініціює компоненти, переналаштовує інші компоненти в залежності від інтерфейсу компонента та його конфігурації.

Компонент управління ресурсами сервісу (КУРС) – компонент, який разом з КУРБП забезпечує ефективне використання відповідних ресурсів..

Компонент логування (КЛ) – компонент сервісного рівня, який відповідає за зберігання та логування всіх подій сервісу для подальшого використання під час аналізу проблем чи малої продуктивності.

Комунікаційний рівень відповідає за формування точок доступу для інших сервісів на базі стандартів WS-Addressing, а також формування запитів до інших сервісів, забезпечуючи їх взаємодію з метою реалізації завдань, що вимагають використання можливостей сукупності сервісів.

Компонент управління надійністю передачі даних (КУНПД) – компонент, який відповідає за підтримку надійної комунікації між бізнес-процесом прикладного рівня та будь-яким іншим сервісом в разі відсутності мережі, проблем з компонентом, якщо це передбачено самим бізнес-процесом під час конфігурації.

Компонент управління безпекою (КУБ) – компонент комунікаційного рівня, який базується на стандартах WS-Security, WS-

Federation, WS-Security: Username Token Profile і забезпечує безпечну взаємодію з сервісами і між сервісами.

Компонент оптимізації трафіку (КОТ) – компонент, який в автоматичному режимі контролює швидкість передачі даних мережею, і за певним алгоритмом може змінювати параметри повідомлень, запитів.

Моделі і методи для реалізації підходу

Моделі планування схеми виконання сервісів. Для підтримки кожного бізнес-процесу чи певної їх системи визначається певна схема виконання програмних модулів підтримки бізнес-процесів. Інформація щодо кожного сервісу, компонента, модуля фіксується в системі (реєстр, інтерфейси та інші джерела) і використовується для створення схеми виконання програмних модулів підтримки бізнес-процесів. Зазначена схема будується на основі опису бізнес-процесу чи їх системи. Реалізовані ручний і автоматичний способи побудови зазначеної схеми. Перший реалізується адміністратором за допомогою зручного графічного інтерфейсу.

Другий базується на логічних моделях, які формалізують процес пошуку відповідно до заданого опису бізнес-процесу потрібних сервісів, компонентів, модулів і побудови структури їх викликів зі специфікацією передачі відповідних параметрів. Таким чином, створюється схема виконання програмних модулів підтримки бізнес-процесів. Відповідний компонент системи відстежує виконання викликів необхідних сервісів і відправляє оброблені дані у відповідності з ланцюжком дій. Це відбувається згідно побудованої схеми виконання програмних модулів підтримки бізнес-процесів.

Опис бізнес-процесу обробляється з урахуванням відомих системі сервісів, їх функціональності, поданих засобами логічного формалізму у вигляді системи аксіом і правил виведення. Сам процес побудови схеми виконання програмних модулів підтримки бізнес-процесів у такому випадку базується на певному методі логічного виведення і механізмі відтворення виводу у вигляді схеми виконання програмних модулів підтримки бізнес-процесів.

Логічний підхід видається прийнятним, оскільки дозволяє вийти на високий рівень вище вимог до інформаційних систем, позбутися недоліків традиційного алгоритмічного підходу, інтегрувати накопичений арсенал сервісів і навіть підключати компоненти, що не створювалися як сервіси. Для описання сервісів будемо використовувати логіку, запропоновану у праці [2]. У тій же праці наведений відповідний метод виведення.

Моделі оптимального використання ресурсів. Для пошуку і ефективного використання вільних ресурсів системи будемо використовувати моделі і методи, запропоновані у праці [5]. Там же наведені моделі управління навантаженням і розподілу ресурсів у разі їх дефіциту.

Особливості реалізації окремих компонентів

На найвищому рівні існують три інтерфейси, які повинні успадкувати компоненти (причому кожен компонент успадковує тільки один з цих трьох інтерфейсів):

IAapplicationLayer – інтерфейс, який представляє можливості доступу до функціональних ресурсів сервісів;

IServiceLayer – інтерфейс, який представляє можливості доступу до функцій управління сервісами;

ITransportLayer – інтерфейс, який представляє можливості доступу до комунікаційних ресурсів сервісів.

Також кожний компонент повинен успадкувати інтерфейс *IComponent <K,T>*, де *K* – вхідний інтерфейс, *T* – вихідний інтерфейс, а також базовий клас *Component*. Кожний із вхідних і вихідних інтерфейсів успадковує від інтерфейсу *IInterServiceCommunication* та базового класу *ServiceChannel*. Клас *ServiceChannel* успадковує від базового класу *BaseChannel*

Власне кажучи, сервіс перетворюють у структурний елемент компоненто-базованого підходу саме WS-компоненти загального використання, до яких належать:

- КУТ, реалізований на базі WS-Coordination, WS-Business Activity, WS-Atomic Transaction;
- КУПВ, реалізований на базі Web Service Choreography Interface;
- КУРБП, реалізований на базі WS-Resource, Web Services Resource Properties, WS-ResourceLifetime, WS-Service Group, WS-BaseFaults;
- КУБ, реалізований на базі WS-Security;
- КУМП, реалізований на базі WS-Context;
- КУРС, реалізований на базі WS-Resource, Web Services Resource Properties, WS-ResourceLifetime, WS-Service Group, WS-BaseFaults;

- КПС, реалізований на базі WS-Discovery.

До головних компонентів загального використання, які реалізують програмні додатки та методи їх виконання, належить компонент Business Process Engine. Він складається з таких модулів:

- верифікації даних на основі DSL-метадекларацій;
- перевірки BPML-документа, завантаженого до компоненту з метою виконання;
- виконання BPML-документа;
- перевірки та передачі стану BPML-документа під час або після виконання;
- завантаження та запуску додатків, створених для .NET Framework.

Модель сервісу, побудованого на базі компонентно-базованого підходу, базується на таких стандартах та специфікаціях:

1) специфікаціях бізнес-процесів:

a) специфікація Business Process Execution Language for Web Services (BPEL4WS) версії 2.0 [1];

b) специфікація XML Process Definition Language (XPDL) версії 2.0;

c) специфікація Business Process Management Language (BPML) версії 1.1 [6];

2) специфікації управління сервісом:

a) специфікація Management Of Web Services (MOWS) версії 1.1 [6];

3) специфікації управління ресурсами:

a) специфікація Web Services Resource Framework (WSRF) версії 1.2 [7]:

i) специфікація Web Services Resource (WS-Resource) версії 1.2 [8];

ii) специфікація Web Services Resource Properties (WS-ResourceProperties) версії 1.2 [9];

iii)специфікація Web Services Resource Lifetime (WS-ResourceLifetime) версії 1.2 [10];

iv)специфікація Web Services Service Group (WS-ServiceGroup) версії 1.2 [11];

v) специфікація Web Services Base Faults (WS-BaseFaults) версії 1.2 [12];

4) специфікації управління передачею повідомлень:

a) специфікація WS-Notification версії 1.3;

b) специфікація SOAP 1.2 [13];

5) специфікації управління безпечної передачі даних:

a) специфікація WS-Security версії 1.1 [14];

6) специфікація управління передачею та обміном міжсервісного контексту - WS-Context (WS-CTX) версії 1.0 [15].

На основі запропонованого підходу реалізовано програмний комплекс розгортання SOA-сервісів для системи компоненто-орієнтованого проектування Component Forest Suite. Його використання для створення і експлуатації інформаційних систем декількох установ підтвердило працездатність закладених рішень. При цьому вдалося до 20% поліпшити деякі параметри інформаційних систем, до яких звичайно висуваються високі вимоги, насамперед максимальний час відповіді і відмовостійкість. Це ж стосується важливої для банківських інформаційних систем легкості підтримки.

Висновки

Запропонований підхід до проектування, реалізації та підтримки сервісів, призначених для використання при створенні, реалізації і експлуатації ІС, побудованих на основі технологій компоненто-базованого проектування. На основі підходу розроблений програмний комплекс, який дозволяє будь-яку бібліотеку на базі .NET Framework розгорнути у вигляді SOA-сервісу з необхідною для реалізації ІС базовою функціональністю. Зазначений комплекс дозволяє здійснювати моніторинг, відстежувати внутрішній стан сервісу, динамічно перезавантажувати та переналаштовувати параметри сервісу та параметри бізнес-процесів, які виконуються за підтримки сервісу. Кожен сервіс може виконувати від одного до N бізнес-процесів в залежності від SLA та конфігурації.

Список посилань

1. Шаховська Н. Б. Сховища та простори даних – інформаційний фундамент систем прийняття рішень / Н. Б. Шаховська, Я. І. Виклюк // Електротехнічні та комп'ютерні системи. – № 08 (84). – 2012. – С. 93-95.
2. Теленик С.Ф. Логічний підхід до інтеграції програмних застосувань підтримки міждисциплінарних наукових досліджень / С.Ф. Теленик, О.А.Амонс, К.В.Єфремов, В.Т.Лиско // Наукові вісті НТУУ «КПІ». – №5 (91). – 2013. – С. 53-72.
3. Теленик С.Ф. Семантична інтеграція різнорідних інформаційних ресурсів / С.Ф. Теленик, О.А.Амонс, К.В.Єфремов, С.В.Жук // Вісник НТУУ «КПІ», Інформатика, управління та обчислювальна техніка. – №58. –2013. – С. 29 - 45.

4. Kikuchi S. Adaptive Integration of Distributed Semantic Web Data / S. Kikuchi, S. Sachdeva, S. Bhalla et al. // Berlin: Databases in Networked Information Systems, Volume 5999, «Springer Berlin Heidelberg», 2010. P. 174-193.
5. Telenyk Sergii. Models and methods of resource management for VPS hosting / Sergii Telenyk, Oleksandr Rolik, Maksym Bukasov, Dmytro Halusko // Technical Transactions series Automatic Control. Vol.4 – AC/2013. – P.41-52.
6. https://www.oasis-open.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf.
7. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf#overview.
8. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.
9. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.
10. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.
11. http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf.
12. http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.
13. https://www.w3.org/TR/soap/.
14. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss#overview.
15. http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf.