УДК 004.94

О.В. Захарова

ВИКОРИСТАННЯ ДЕСКРИПТИВНИХ ЛОГІК В ПРОБЛЕМАТИЦІ WEB-СЕРВІСІВ

На сьогодні Web-сервіси дозволяють вирішувати конкретні бізнес-задачі, що реалізують бізнес процеси у різних галузях життєдіяльності людини. Але для того, щоб отримати виконуваний Web-сервіс, треба вміти ефективно вирішувати цілу низьку задач самих Web-сервісів на всіх етапах їх життєвого циклу. Апарат дескриптивних логік, завдяки своїм механізмам міркувань та можливостям логічного виводу та надання описам семантичного змісту, є ефективним та потужним інструментом для вирішення задач Web-сервісів. Мета даної роботи полягає у визначенні ланцюжка задач Web-сервісів на функціональному рівні та підходів до вирішення цих задач із застосуванням апарата дескриптивних логік.

Вступ

На сьогодні Web-сервіси – це програмні компоненти, що дозволяють вирішувати конкретні бізнес-задачі, реалізуючі відповідні бізнес-процеси. Як правило, неможливо знайти один Web-сервіс, що реалізує бізнес-процес. Є необхідність у побудові складних – композитних Web-сервісів, поєднуючі існуючі сервіси у крупніші системи. Зрозуміло, що створення такого сервісу має здійснюватися автоматизованим чином. А ефективність вирішення цієї задачі та вибору тих чи інших алгоритмів на кожному з етапів життєвого циклу сервісу, перш за все, залежить від обраної моделі формалізації сервісу.

Потрібний формалізм, що забезпечуватиме хороше семантичне анотування сервісів та підходи для семантичного співставлення сервісів для автоматизованого виявлення підходящого сервісу та побудови композитного сервісу, який реалізує конкретний бізнес-процес. Так як сервіси можуть створювати змінення в світі, точне представлення їх функціональності повинно мати справу з динамічним аспектом (поведінкою).

Це є одним з ключових аспектів, що робить очевидною доцільність використання дескриптивних логік (ДЛ) для описання сервісів, так як ДЛ забезпечує можливість міркувань для сервісів, і таким чином дозволяє описати їх динамічний аспект.

Окрім цього, було б доцільним, щоб формалізм базувався на онтології задач, які реалізуються сервісом. А ДЛ лежать в основі таких широко відомих мов описання онтологій, як OWL та його модифікації.

Визначення сервісу

Сервіс-орієнтована архітектура (SOA) з кожним днем набирає сили як нова модель організації взаємодії різноманітних корпоративних прикладних систем. Але однією з суттєвих проблем цієї галузі є відсутність чітких стандартів та нестача інформації про сервіси при стрімкому зростанні кількості SOA- розробок. У галузі розробки та прийняття стандартів та специфікацій для Web-сервісів можна виділити три основні організації: WS-I (Web Services Interoperability Organization), W3C (World Wide Web Consortium) та OASIS (Organization for the Advancement of Structured Information Standards).

Згідно визначення W3C [1], під *сервісом* розуміють таку програмну систему, що ідентифікується URI, та публічні інтерфейси та прив'язки якої визначені та описані мовою XML. Опис цієї програмної системи може бути знайдено іншими програмними системами, які можуть взаємодіяти з нею відповідно до цього опису з використанням повідомлень, що базуються на XML та передаються за допомогою Інтернет-Протоколів. Web-сервіси базуються на чотирьох ключових технологіях [1–25]: eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) та Universal Description, Discovery and Integration (UDDI). Ці технології використ-

товуються для забезпечення функціонування Web-сервісів.

XML – розширювана мова розмітки, призначена для зберігання і передачі структурованих даних, є основою для таких технологій, як SOAP та WSDL. Визначає формат даних, що використовується для обміну інформацією між споживачем сервісу та самим сервісом.

SOAP (Simple Object Access Protocol, або Services-Oriented Architecture Protocol) – заснований на мові XML стандарт для взаємодії між сервісами та їх споживачами.

WSDL – це мова опису зовнішніх інтерфейсів Web-служби на базі XML. Описаний таким чином інтерфейс містить всю інформацію, яка необхідна для доступу до даного сервісу. З точки зору WSDL-документа Web-сервіс являє собою колекцію портів, які, в свою чергу, є колекцією абстрактних операцій та повідомлень. Абстракція операцій та повідомлень дозволяє зв'язувати їх з різними протоколами та форматами даних типу SOAP, HTTP GET/POST або MIME.

UDDI (Universal Discovery, Description, and Integration) – універсальний інтерфейс розпізнавання опису та інтеграції. Задача UDDI – надати механізм для публікації інформації про Web-сервіси, а також забезпечити підтримку пошуку Web-сервісів, що є доступними. В цілому UDDI – це регістраційна система, до якої входять набір XML-файлів та асоційовані схеми, що містять описи послуг, які надаються Web-сервісами.

Зв'язки між цими стандартами показані на рис. 1.

Базовими ролями сервіс-орієнтованої архітектури є провайдер сервісу, запитувач сервісу та механізм виявлення [17]:

- провайдер сервісу – будь-яка організація, що надає сервіс, така сутність забезпечує специфічну реалізацію сервісу;
- запитувач сервісу (клієнт) – сутність, яка шукає сервіс та запускає конкретний сервіс, щоб досягти своїх цілей;
- механізм виявлення діє як репозиторій або довідник сервісів (реєстр).

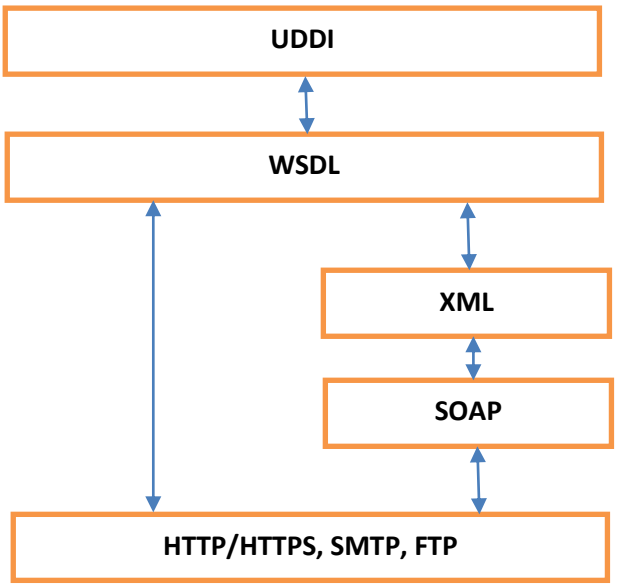

Рис. 1

Постановка задачі

Швидкий розвиток сервіс-орієнтованих технологій, можливості повторного використання Web-сервісів та реалізація прикладних бізнес-систем як сервісів, що існують в Інтернет, величезна кількість доступних сервісів виводить на передній план задачу створення такого механізму, що б дозволив автоматично знайти коректний Web-сервіс, який задовольнятиме вимоги користувача. Але, скоріше за все, може не існувати такого окремого Web-сервісу і тоді потрібна можливість сформувати вибірку існуючих сервісів та інтегрувати їх таким чином, щоб задовольнити функціональність користувача. Така можливість називається композицією Web-сервісів. Результат композиції має генеруватися автоматично при інтерактивній взаємодії з користувачем на базі існуючих сервісів та цільового запиту. На сьогодні процес інтеграції містить багато ручної праці. Враховуючи велику кількість сервісів, які треба інтегрувати для реалізації функціональності найпростішої прикладної системи, питання автоматизації процесу побудови композиції стає критичним для забезпечення масштабованості сервіс-орієнтованих систем. Тобто необхідно передати функції людини машині. Для цього, перш за все, треба доповнити моделі сервісів семантикою,

яка може бути потім використана механізмами міркувань при вирішенні різних задач життєвого циклу сервісів. Необхідність реалізації автоматизованого логічного виведення обумовлює доцільність використання в алгоритмах вирішення задач Web-сервісів дескриптивних логік, що мають готові, та вже досить розвинені механізми міркувань.

Дескриптивні логіки

Дескриптивні логіки (ДЛ) [2] – це родина мов представлення знань, що може використовуватися для представлення знань прикладного домена структурованим та формально добре зрозумілим способом. Назва дескриптивні (описові) логіки мотивується тим фактом, що з одного боку важливі поняття домена описуються дескрипторами концептів, тобто виразами, які будуються з атомних концептів (унарних предикатів) та атомних ролей (бінарних предикатів), використовуючи конструктори ролей та концептів, які забезпечуються конкретною ДЛ. Множина цих конструкторів й визначає виразну потужність ДЛ.

Мінімальний набір конструкторів наведений у табл. 1 далі [3] (за основу береться $\mathcal{ALCQIO}$ – базова ДЛ $\mathcal{ALC}$, що розширена номіналами та інверсними ролями).

Таблиця 1. Синтаксис та семантика $\mathcal{ALCQIO}$

Назва	Синтаксис	Семантика
Інверсна роль	s^-	$\{(y,x) \mid (x,y) \in s^{\mathcal{J}}\}$
Кон'юнкція	$C \sqcap D$	$C^{\mathcal{J}} \cap D^{\mathcal{J}}$
Доповнення	$\neg C$	$\Delta^{\mathcal{J}} / C^{\mathcal{J}}$
Обмеження «щонайменше»	$(\geq nrC)$	$\{x \mid card(\{y \mid (x,y) \in r^{\mathcal{J}} \wedge y \in C^{\mathcal{J}}\}) \geq n\}$
Номінали	$\{a\}$	$\{a^{\mathcal{J}}\}$

За допомогою цих конструкторів може бути визначений ряд інших конструкторів, як абревіатура: диз'юнкція ($\sqcup$), верхній концепт ($\top$), обмеження «щонайбільше» ($\leq nrC$), обмеження існування ($\exists r.C$).

Семантики концептів та ролей ДЛ визначаються в термінах ***Інтерпретації*** [18]. Це модель, що формально складається з непорожньої множини $\Delta^{\mathcal{J}}$ (домена) та функції інтерпретації $\cdot^{\mathcal{J}}$, яка співставляє кожному атомарному концепту A дескриптивної логіки деяку підмножину $A^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}}$, а кожній атомарній ролі R — деяку підмножину $R^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$, кожному індивіду a – елемент $a^{\mathcal{J}} \in \Delta^{\mathcal{J}}$.

ДЛ відрізняються від своїх попередників, таких як семантичні мережі та фрейми, тим, що вони наділені формальними семантиками, що засновані на логіці.

Завдяки тому, що:

- у ДЛ знання доступні для машинної обробки – автоматизованого логічного виводу нових знань з наявних;
- мова має точну семантику, а логічні проблеми є вирішувальними (мають практично допустиму обчислювальну складність);
- мова має значну виразну силу, що пригодна для формулювання на цій мові практично всіх значимих фактів.

ДЛ стали основою декількох мов Web-онтологій.

База знань ДЛ складається з інтенсіональних знань у вигляді термінології (TBox) та екстенсіональних знань (assertional), що специфічні для екземплярів домена (ABox). TBox будується за допомогою оголошень, що описують загальні властивості понять, та, як правило, не змінюються, тому вважається статичним. Знання ABox часто змінюються.

Якщо жоден концепт TBox не визначається сам через себе, тобто його визначення не містить його ім'я напрямку, або побічно, то такий TBox називається ***ациклічним*** [14].

Системи ДЛ забезпечують користувачів різними механізмами виводу, що виводять неявні знання з тих, що явно представлені. Більш того, на сьогодні існує

вже чимало реалізованих механізмів міркувань ДЛ, що готові до використання. Всі ці переваги обумовлюють безперечну доцільність використання апарату ДЛ для вирішення багатьох задач Web-сервісів. Але, тут варто пам'ятати, що окремою задачею залишається вибір такої ДЛ, що є компромісним рішенням між її виразністю та розв'язуваністю. Ця проблема потребує уваги, але залишається за межами даної роботи. Для демонстрації прикладів цієї роботи досить можливостей $\mathcal{ALCQIO}$.

Основні задачі життєвого циклу сервісу на функціональному рівні

На практиці існує два рівні представлення сервісів, а саме: функціональна та процесна модель сервісів [17]. На функціональному рівні сервіси розглядаються як окремі існуючі в мережі прикладні компоненти, які можуть бути викликані шляхом відправлення повідомлення. При цьому сервіс виконує свою задачу та (в деяких випадках) виробляє відповідь тому, хто його викликав. Таким чином, відсутня безперервна взаємодія між запитувачем сервісу та самим сервісом. Опис таких Web-сервісів фокусується на їх функціональності в термінах імені сервісу, імен операцій, імен повідомлень (що відомі також як вхідні та вихідні повідомлення /параметри), імені інтерфейсу.

Сервіси процесного рівня містять декілька операцій, які слідують загальній поведінці сервісу. Такі сервіси потребують розширеної взаємодії між запитувачем сервісу та множиною операцій, забезпечуючи конкретну функціональність. Таким чином, це, як правило, композитні сервіси, тобто взаємодія з ними складається не лише з окремого кроку запит-відповідь, для досягнення потрібного результату вони мають слідувати складному протоколу. Ці кроки можуть складатися з довільних (умовних та ітеративних) комбінацій з умовними виходами. На цьому рівні необхідний деталізований опис внутрішньої поведінки сервісів, наприклад, за допомогою (STS). Очевидно, що на процесному рівні Web-сервіси також мають функціональний опис своїх операцій та сервісів. Але сервіси на функціональному рівні не мають поведінкових взаємодій. Це є головною відмінністю цих двох моделей.

Предметом розгляду в цій роботі є лише функціональна модель сервісу.

Як ми вже зазначили, ***функціональна модель сервісу*** [15] описує його функціональність та визначає способи, як клієнт може взаємодіяти з сервісом для досягнення його функціональних можливостей. Це робиться шляхом вираження перетворення даних з входів і виходів і перетворення стану з передумов і ефектів. Тобто сервіс розглядається як атомний елемент, а композитний сервіс як пов'язана сукупність атомних елементів. Процес, що відбувається всередині сервісу залишається поза увагою. Вирішення задач Web-сервісів з урахуванням їх процедурної поведінки виходить за межі цієї статті.

Можна визначити наступні базові задачі життєвого циклу сервісу на функціональному рівні, для вирішення яких доцільне використання апарату ДЛ:

- формалізація Web-сервісу на функціональному рівні. Передумови та ефекти сервісу задаються як твердження ДЛ;
- виявлення Web-сервісу:

а) визначення цілі виявлення як кон'юнктивного запиту, що є кон'юнкцією тверджень ДЛ;

б) співставлення (matching) пошукового запиту та сервісів на функціональному рівні, перевірка відповідності передумов та ефектів цілі виявлення (goal discovery), перевірка істинності передумов;

в) верифікація Web-сервісу – перевірка відповідності сервісу його специфікації за допомогою резонерів ДЛ;

г) перевірка не функціональних характеристик сервісу [18, 25], таких як вартість, повноваження, ступінь довіри до сервісу, репутація сервісу та інші. Слід зазначити, що врахування не функціональних характеристик сервісу підвищує якість відбору сервісів, але суттєво ускладнює задачу;

- побудова композитного сервісу на функціональному рівні – перевірка відповідності між передумовами та ефектами складових сервісів.

Опис Web-сервісу на функціональному рівні

На цьому рівні надаються можливості опису сервісу у термінах вхідних та вихідних параметрів, передумов (умов виконання сервісу) та ефектів (результатів виконання сервісу).

Вхідні параметри – це те, що потребує сервіс, щоб виробити бажаний результат. ***Передумови*** представляють умови у світі, які мають виконуватися, тобто бути істинними, для успішного виконання сервісу. Виконання сервісу може мати результат у діях в світі. Ці умови описуються як ***ефекти*** сервісу. В результаті можуть бути встановлені значення ***вихідних параметрів*** сервісу, які надалі можуть бути використані в задачі, наприклад, як значення вхідних параметрів для іншого сервісу.

За типами вхідних параметрів розрізняють два види сервісів: базові та параметричні [3].

Під ***базовими*** розуміють сервіси, вхідні параметри яких вже встановлені іменами індивідів.

Параметричні сервіси містять на місці імен індивідів змінні, та повинні розглядатися як компактне представлення всіх їх базових (ground) екземплярів. Для простоти далі розглядаються лише базові сервіси, вважаючи, що параметричні сервіси вже були визначені екземплярами.

Профілі сервісу описують як самі сервіси, що надаються, так і запити до сервісів. Запит складається з опису гіпотетичного сервісу, що виконує задачу, яка потрібна особі, що запитує сервіс.

Виходячи з вищесказаного та враховуючи існуючі визначення сервісу [3, 22], можна формально описати сервіс на базі формалізму ДЛ наступним чином.

Нехай $\mathcal{T}$ – ациклічний TBox. *Атомний сервіс* $S = (in, out, pre, post)$ для ациклічного TBox $\mathcal{T}$ складається з

1) кінцевої множини *pre* ABox чи TBox тверджень – передумов, типу $A(a)$, $s(a; b)$, *де* A – ім'я примітивного концепта в $\mathcal{T}$ а s – ім'я ролі;

2) кінцевої множини *in* вхідних параметрів;

3) кінцевої множини *post* умовних постумов (або ефектів) форми φ/ψ, де φ – це твердження ABox і воно є примітивним літералом для $\mathcal{T}$, тобто, твердження ABox $A(a)$, $\neg A(a)$, $s(a; b)$, або $\neg s(a; b)$ з ім'ям примітивного концепта A в $\mathcal{T}$ та ім'ям ролі s. Умовні пост умови φ/ψ кажуть, що, якщо φ було істинно до виконання сервісу, то ψ має бути істинним після;

4) кінцевої множини *out* вихідних параметрів.

Інтуїтивно, сервіс є відношенням на інтерпретаціях. Тобто, сервіс S зв'язує інтерпретацію $\mathcal{J}$ з інтерпретацією $\mathcal{J}$' при виконанні наступного твердження: якщо в результаті виконання сервісу S можна отримати інтерпретацію $\mathcal{J}$', то цей сервіс може бути застосований в $\mathcal{J}$. А саме, це означає, що виконання сервісу S переводить інтерпретацію $\mathcal{J}$ в $\mathcal{J}$'. Це можливо лише, якщо $\mathcal{J}$ та $\mathcal{J}$' задовольняють предта пост- умови сервісу S [19]:

- умови, що мають задовольнятися для виконання S, описуються у pre. Умова $\forall A$ вимагає, щоб кожний індивід в $\mathcal{J}$ був екземпляром концепта A. Решта умов відповідають твердженням ABox ДЛ [2];
- умови, що мають задовольнятися після виконання S (тобто інтерпретацією $\mathcal{J}$'), описуються в $post$. Так як стан світу до S може впливати на ефекти S, пост-умови трохи відрізняються від предумов. Ідея, що стоїть за кожним φ/ψ, полягає у тому, що, якщо кожна умова в φ міститься в $\mathcal{J}$, то ψ має міститися в $\mathcal{J}$'.

Більше того, бажано, щоб $\mathcal{J}$' мінімально відрізнялася від $\mathcal{J}$, тобто лише в тих аспектах, які вимагаються постумовами. Це забезпечується семантиками сервісу та є сутністю підходу мінімізації змін.

Інтерпретації відрізняються одна від одної, якщо існує концепт або роль, інтерпретація якого(ої) присутня в одній моделі, але відсутня в іншій. Формально це можна визначити наступним чином.

Визначення 1 [19]. Дві інтерпретації $\mathcal{J}$ та $\mathcal{J}$' відрізняються відносно $\boldsymbol{\mathcal{T}}$ ($\mathcal{J} \neq \mathcal{J}$') в $d \in \Delta^{\mathcal{J}}$ та імені концепта C_p, якщо $d \in C_p^{\mathcal{J}}$ та $d \notin C_p^{\mathcal{J}'}$ або $d \notin C_p^{\mathcal{J}}$ та $d \in C_p^{\mathcal{J}}$. Аналогічно, дві інтерпретації $\mathcal{J}$ та $\mathcal{J}$' відрізняються відносно $\boldsymbol{\mathcal{T}}$ в $< d, e > \in \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$ та імені ролі R, якщо $< d,e > \in R^{\mathcal{J}}$ та $< d,e > \notin R^{\mathcal{J}'}$ або $< d,e > \notin R^{\mathcal{J}}$ та $< d,e > \in R^{\mathcal{J}'}$. У даному визначенні d або $< d,e >$, відповідно, утворюють симетричну різність інтерпретацій.

В роботі [6] визначається відношення пріоритетності $\preccurlyeq_{\mathcal{J}S\mathcal{T}}$ на інтерпретаціях, що характеризує їх "близькість" до заданої (вихідної) інтерпретації $\mathcal{J}$. Тоді, інтерпретацію, якій надається перевага, можна визначити наступним чином.

Визначення 2 [3]. Нехай $\boldsymbol{\mathcal{T}}$ – ациклічний TBox, $S = (pre; post)$ – сервіс для $\boldsymbol{\mathcal{T}}$, та $\mathcal{J}$ – модель для $\boldsymbol{\mathcal{T}}$. Бінарне відношення $\preccurlyeq_{\mathcal{J}S\mathcal{T}}$ на моделях $\boldsymbol{\mathcal{T}}$, визначається, як $\mathcal{J}$'

$$\preccurlyeq_{\mathcal{J}S\mathcal{T}} \mathcal{J}'', \text{ якщо } \mathcal{J}' \neq \mathcal{J}''$$
$$A^{\mathcal{J}} \nabla A^{\mathcal{J}'} \subseteq A^{\mathcal{J}} \nabla A^{\mathcal{J}''};$$
$$s^{\mathcal{J}} \nabla s^{\mathcal{J}'} \subseteq s^{\mathcal{J}} \nabla s^{\mathcal{J}''},$$

де ∇ позначає симетричну різність між двома множинами.

Виходячи з цього, застосування сервісу можна визначити наступним чином.

Визначення 3 [3]. Нехай $\boldsymbol{\mathcal{T}}$ – ациклічний TBox, $S = (pre; post)$ – сервіс для $\boldsymbol{\mathcal{T}}$, та моделей $\boldsymbol{\mathcal{T}}$ - $\mathcal{J}$,$\mathcal{J}$', які розділяють той самий домен та інтерпретацію всіх імен індивідів. S може перетворювати $\mathcal{J}$ в $\mathcal{J}$' ($\mathcal{J} \Rightarrow_S^{\mathcal{T}} \mathcal{J}'$) лише, коли $\mathcal{J}$,$\mathcal{J}' \models$ post, та не існує моделі $\mathcal{G}$ TBox $\boldsymbol{\mathcal{T}}$ такої, що $\mathcal{J}$, $\mathcal{G} \models$ post, $\mathcal{G} \neq \mathcal{J}'$, та $\mathcal{G} \preccurlyeq_{\mathcal{J}S\mathcal{T}} \mathcal{J}'$.

Аналогічно, композитний сервіс $S_1,\ldots,S_k$ може перетворювати $\mathcal{J}$ в $\mathcal{J}'$ ($\mathcal{J} \Rightarrow_{S_1,\ldots S_k}^{\mathcal{T}} \mathcal{J}'$) лише тоді, якщо існують моделі $\mathcal{J}_0, \ldots, \mathcal{J}_k$ TBox $\mathcal{T}$ з $\mathcal{J}=\mathcal{J}_0$, $\mathcal{J}'\mathcal{J}_0=\mathcal{J}_k$, та $\mathcal{J}_{i-1} \Rightarrow_{S_i}^{\mathcal{T}} \mathcal{J}_i$ для $1 \leq i \leq k$.

Але нав'язування цієї мінімізації змін є занадто обмеженою [4, 5], і не для всіх примітивних літералів може бути застосована умова мінімізації. Тому, для вирішення цієї проблеми, а саме для опису таких літералів, ряд підходів до формалізацій сервісу, пропонують використовувати окклюзії [3] – перепони. Тобто до складових визначення додається кінцева множина *occ* окклюзій форми $A(a)$ або $r(a; b)$, де A примітивне ім'я концепта з $\boldsymbol{\mathcal{T}}$, r – ім'я ролі, та $a; b$ – індивіди.

Також, слід зазначити, що множина вхідних параметрів *In*, може розглядатися як частина передумов сервісу, тому що вимога до наявності того чи іншого вхідного параметра також є умовою виконання сервісу – найпростіша умова. Аналогічно, множина вихідних параметрів *Out* може розглядатися як частина множини постумов, де φ – це просто тавтологія, а ефектом є формування значення вихідного параметра.

Композитний сервіс для $\boldsymbol{\mathcal{T}}$ визначається як кінцева послідовність $S_1,\ldots,S_k$ атомних сервісів для $\boldsymbol{\mathcal{T}}$.

Як приклад визначення сервісів, розглянемо простий сервіс продажу велосипедів [19]:

$$(\{owns(a_1, b),\ wants(a_2, b), owns(a_2,\ p),\ Bicycle(b)\},\ \{\varnothing / owns(a_2, b),\ \varnothing / owns(a_1,\ p)\}). \quad (1)$$

Тобто множина передумов сервісу: $pre = \{owns(a_1, b),\ wants(a_2, b), owns(a_2,\ p),\ Bicycle(b)\}$, множина умовних пост-умов: $post = \{\varnothing / owns(a_2, b), \varnothing / owns(a_1,\ p)\}$, де $\varnothing$ – порожня множина умов в умовній пост-умові. Порожня множина задовольняється кожною інтерпретацією.

У наведеному прикладі – функціональна роль (або база знань, що містить аксіоми типу $\top \sqsubseteq (\leq 1\ owns^{-})$), вартість велосипеда моделюється за допомогою абстрактного об'єкта p, та нічого більше не має змінюватися при купівлі/продажу велосипеда. Такий простий сервіс описує продаж велосипеда відповідним чином: завдяки семантикам, лише власник велосипеда та його винагорода змінюються сервісом. Так, наприклад, b залишається велосипедом. Якщо велосипед виявився поганим, новий власник буде не задоволений, а

колишній – навпаки задоволений. Це можна змоделювати наступним чином:

$$(\{owns(a_1, b), wants(a_2, b), owns(a_2, p), Bicycle(b)\}, \{\varnothing / owns(a_2, b), \{Bad(b) / \neg Happy(a_2)\} \varnothing / owns(a_1, p), \{Bad(b) / Happy(a_1)\}\}). \quad (2)$$

Зазвичай ми не маємо повної інформації про світ, тобто модель $\mathcal{J}$ Tbox $\boldsymbol{\mathcal{T}}$ не відома повністю. Нам відомі лише деякі факти, тобто ми маємо ABox $\mathcal{A}$, та разом з $\boldsymbol{\mathcal{T}}$ розглядаються всі моделі $\boldsymbol{\mathcal{A}}$, як можливі стани світу. Перш ніж використовувати сервіс, треба знати, чи може він бути застосованим, тобто, чи задовольняються всі передумови. Якщо так, можливо треба знати, чи досягає його застосування бажаного ефекту, тобто, чи дійсно твердження, яке ми бажаємо зробити правильним, зберігається після виконання сервісу. Ці проблеми є базовими задачами виводу в ДЛ [7], та в наведеній вище постановці можуть формально визначатися наступним чином.

Визначення 4. Нехай $S_1,\ldots,S_k$ – сервіс для ациклічного TBox $\boldsymbol{\mathcal{T}}$, з $S_i = (pre_i; post_i)$ та A - ABox.

- *Виконуваність*: сервіс $S_1,\ldots,S_k$ є виконуваним в $\boldsymbol{\mathcal{A}}$ відносно $\boldsymbol{\mathcal{T}}$ лише тоді, якщо наступні умови вірні у всіх моделях $\mathcal{J}$ $\boldsymbol{\mathcal{A}}$ та $\boldsymbol{\mathcal{T}}$:
  - $\mathcal{J} \vDash pre_1$ та
  - для всіх i з $1 \le i \le k$ та всіх інтерпретацій $\mathcal{J}'$ з $\mathcal{J} \Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_k} \mathcal{J}'$, маємо $\mathcal{J}' \vDash pre_{i+1}$.
- *Проекція*: твердження φ є послідовністю застосування $S_1,\ldots,S_k$ в $\boldsymbol{\mathcal{A}}$ відносно $\boldsymbol{\mathcal{T}}$ лише тоді, якщо для всіх моделей $\mathcal{J}$ $\boldsymbol{\mathcal{A}}$ та $\boldsymbol{\mathcal{T}}$, та всіх $\mathcal{J}'$ з $\mathcal{J} \Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_k} \mathcal{J}'$, маємо $\mathcal{J}' \vDash \varphi$.

Таким чином, щоб досягти ефекту φ (твердження ABox), сервіси $S_1,\ldots,S_k$ мають бути виконуваними в $\boldsymbol{\mathcal{A}}$ відносно $\boldsymbol{\mathcal{T}}$, кожний S_i має бути погодженим з $\boldsymbol{\mathcal{T}}$ для $1 \le i \le k$, та φ має бути послідовністю застосування $S_1,\ldots,S_k$ в $\boldsymbol{\mathcal{A}}$ відносно $\boldsymbol{\mathcal{T}}$.

Виявлення сервісів на функціональному рівні

Щоб знайти Web-сервіси для виконання конкретних задач у бізнес-процесі, запитувач має, перш за все, визначити умови, які має задовольняти цей Web-сервіс. Такий підхід називається співставлення на основі цілі (goal based matchmaking) [8, 9] та має справу з визначенням умов на оголошення Web-сервісу (вхідні, вихідні параметри, передумови та ефекти) та перевіркою, чи може Web-сервіс задовольняти ці умови. Цілі, як правило, слідують з бізнес-цілей, та можуть виводитись з них автоматично або не автоматично.

Опис чи оголошення сервісу відповідає запиту, якщо запит є досить близьким до сервісу, що запитується. Тут необхідно специфікувати, що означає «досить близький». У найсуворішій інтерпретації, оголошення та запит є «досить близькими», якщо вони описують точно один й той самий сервіс. Але це визначення є занадто обмеженим. Потрібно більш гнучке визначення поняття «досить близьке». Тобто, необхідні механізми співставлення, які розпізнають ступінь подібності між оголошеннями сервісів та запитами. А в того, хто запитує сервіс, має бути можливість визначати ступінь гнучкості, яку вони надають системі. Чим менша гнучкість, тим менша вірогідність знаходження сервісів, що задовольняють вимогам.

Таким чином:

- механізм співставлення має підтримувати гнучке семантичне співставлення між оголошеннями сервісів та запитами на основі онтологій, що доступні сервісам та механізму співставлення;
- запитувач має володіти деяким контролем ступеня гнучкості співставлення, що дозволяється системі;
- механізм співставлення має заохочувати обидві сторони бути чесними у своїх описах у питаннях вартості;
- процес співставлення має бути ефективним: він не має навантажувати запитувача надмірними затримками, які перешкоджатимуть його ефективності.

Далі наводиться алгоритм, який адоптує множину стратегій. Його зміст

полягає у тому, що оголошення сервісу, який шукають, співставляють із запитом. Слід зазначити, що запит також є своєрідним оголошенням сервісу, а саме, того сервісу, який реалізує потрібну користувачеві функціональність. Таким чином, оголошення бажаного сервісу співставляється з оголошеннями сервісів, що існують у реєстрі, і відношення між запитом та оголошенням сервісу ідентичні відношенням між оголошеннями двох сервісів.

Вважаємо, що оголошення сервісу відповідає запиту, якщо всі виходи запиту співпадають з виходами оголошення, а всі входи оголошення співпадають зі входами запиту. Точне співпадання знайти досить важко, тому визначають ступені відповідності і обирають сервіси з найбільшим ступенем відповідності. Базовий критерій виявлення полягає у тому, що відповідний сервіс має задовольняти потреби запитувача, а запитувач має забезпечувати відповідному сервісу всі входи, які необхідні для його коректного функціонування.

Критерій відповідності. Якщо $\mathcal{T}$ – ациклічний Tbox ДЛ $\mathcal{L}$, $S = (In_s, Out_s)$ – сервіс та $Q = (In_q, Out_q)$ – запит, де:

- In_q – кінцева множина входів запиту;
- In_s – кінцева множина входів оголошення сервісу;
- Out_q – кінцева множина виходів запиту;
- Out_s – кінцева множина виходів оголошення сервісу, що виражені твердженнями $\mathcal{L}$, то відповідність сервісу запиту гарантується включеннями $In_s \sqsubseteq In_q$ та $Out_q \sqsubseteq Out_s$. Таким чином, сервіс відповідає запиту, якщо запит містить всі вхідні твердження сервісу та, можливо, ще додаткові, а множина вихідних тверджень запиту, навпаки, має бути вужча за множину вихідних тверджень сервісу, тобто сервіс може повертати більше ефектів ніж того потребує запит.

Запити співставляються за цим критерієм з усіма оголошеннями сервісів, які є у реєстрі. Як тільки знайдена відповідність запиту та оголошення, вона записується, щоб знайти відповідність більш високого ступеня. Ступінь успіху залежить від ступеня виявленого співпадання. Ступінь відповідності між двома входами або двома виходами залежить від відношення між концептами, які пов'язані з цими входами або виходами, а саме, мінімальною відстанню між цими поняттями у дереві таксономії. Розрізняють наступні ступені відповідності [10, 20]:

- *Exact* – точне співпадання:
  - $Out_q = Out_s$ – еквівалентність, або
  - Out_q *subclassOf* Out_s – результат точний у припущенні, що оголошуючи Out_s провайдер сервісу має забезпечити виходи, що погоджені з кожним безпосереднім підтипом Out_s;
- *Plug In* – більш слабкий зв'язок (Out_s *subsumes* Out_q). Якщо Out_s *subsumes* Out_q, то Out_s – це множина, яка включає виходи Out_q, або, іншими словами, Out_s може бути підключений замість Out_q;
- *Subsumes* (Out_q *subsumes* Out_s) – якщо Out_q *subsumes* Out_s, то провайдер не повністю задовольняє запит. Це означає, що запитувач може використовувати провайдера для досягнення своєї мети, але ймовірно йому прийдеться модифікувати свій план або виконати інші запити, щоб завершити свою задачу;
- *Fail* – невдача відбувається, якщо не знайдено будь-якого *subsumption* між Out_q та Out_s.

Ступінь відповідності відображається на дискретній шкалі, де найбільш переважним є точна відповідність (*Exact*), наступний рівень – *Plug In*, так як вихід, що повертається, може бути використаний замість того, що очікує запитувач. *Subsumes* – третій рівень, так як вимоги запитувача виконуються лише частково: оголошений сервіс може забезпечити лише деякі конкретні варіанти того, що бажає запитувач. Самий низький рівень *Fail* – не прийнятний результат. Далі сервіси сортируються за ступенем відповідності за-

питу: обираються сервіси з найбільшим ступенем відповідності виходів. Співставлення входів використовується лише як допоміжна (вторинна) оцінка, щоб розмежувати еквівалентно оцінені виходи. Запитувач може вирішити будь-які невідповідності між доступною інформацією та очікуваннями провайдера шляхом вирішення додаткової задачі або шляхом запиту до реєстру, щоб знайти додаткових провайдерів. Слід зазначити, що можливі варіанти, коли оголошення сервісу та запиту поверхньо виглядають різними, але вони можуть при цьому точно співпадати на основі онтологічних визначень.

Продемонструємо відношення між запитом та оголошеннями сервісів на вище наведеному прикладі сервісів (1) та (2).

$Q = (\{owns(a_1, b),\ wants(a_2, b), owns(a_2, p),\ Bicycle(b)\},$
$\{\varnothing / owns(a_2, b),\ \varnothing / owns(a_1, p),$
$\{\varnothing / Happy(a_2)\})$

$S_1 = (\{owns(a_1, b),\ wants(a_2, b), owns(a_2, p),\ Bicycle(b)\},$
$\{\varnothing / owns(a_2, b),\ \varnothing / owns(a_1, p)\})$

$S_2 = (\{owns(a_1, b),\ wants(a_2, b), owns(a_2, p),\ Bicycle(b)\},$
$\{\varnothing / owns(a_2, b),\ \{Bad(b) / \neg Happy(a_2)\}$
$\varnothing / owns(a_1, p),\ \{Bad(b) / Happy(a_1)\}\}).$

Очевидно, що в цьому прикладі Out_Q *subsumes* Out_{S_1}, тобто сервіс S_1 не повністю задовольняє запит, водночас, як результат співставлення сервісу S_2 с запитом Q буде *Fail*.

Механізм співставлення забезпечує гнучку семантичну відповідність між оголошеннями та запитом. Єдиним питанням у ході співставлення є – чи може цей алгоритм провести виведення між входами та виходами оголошень та запитів на основі онтологій, що доступні реєстру. Більше того, результат не є строго правильним або не правильним, він залежить від ступеня подібності понять у співставленні. Не зважаючи на таку гнучкість, алгоритм співставлення відсікає оголошення, що не відповідають запиту, та приймає, але з низьким рейтингом, відповідності, які можуть бути незадовільними для запитувачів. Запитувач може визначити свої побажання щодо типів відповідностей, які мають виконувати алгоритм співставлення, обмежуючи при цьому мінімальний ступінь відповідності. Також може бути обмежена кількість пошуків, примушуючи алгоритм співставлення обмежити пошук в замкненій підмножині концептів онтології.

Побудова композитного сервісу на основі Web-сервісів, які описані за допомогою ДЛ, на функціональному рівні

Коли розглядається композиція сервісів, треба приймати до уваги зв'язки між різними Web-сервісами. Ці відношення можна класифікувати.

Визначення 5 [11]. Нехай $\boldsymbol{\mathcal{T}}$ – ациклічний Tbox, $\boldsymbol{\mathcal{A}}$ - ABox, та сервіси S_i та S_j є підсервісами композитного сервіса S, тоді як сервіс S_i забезпечує інший тип сервіса, ніж сервіс S_j. Зв'язок R між підсервісами S_i та S_j може бути визначений наступним чином:

- *незалежність:* $S_i + S_j = S_j + S_i$ – кожний підсервіс є вільно незалежним від іншого та порядок їх виконання не має результатом композитний сервіс;
- *ідентичність:* $S_i = S_j$ – два сервіси забезпечують одну й ту саму послугу, але вони мають деякі різні атрибути;
- *умовна ідентичність*: $S_i \cong S_j$ – S_i може забезпечувати ту саму функцію, що й S_j у деякій ситуації;
- *підстановка* (Substitute): $S_i \prec S_j$ – сервіс S_i може заміщуватися сервісом S_j у будь-якому випадку. Аналогічно, $S_j \prec S_i$;
- *умовна підстановка:* $S_i \preccurlyeq S_j$ – сервіс S_i може бути заміщений сервісом S_j в деякій ситуації. Аналогічно, $S_i \succcurlyeq S_j$;
- *перекриття:* $S_i \otimes S_j$ – існує частина перекриття між цими двома сервіса-

ми. Для створення цього відношення, має бути виключена частина, що є перекриттям;

- *передумова:* $S_i \to S_j$ – один сервіс має завершитися до початку другого. Сервіс S_i має завершитися до того, як почнеться сервіс S_j.

Виходячи з цього визначення, сервіси S_1 (1) та S_2 (2) є умовно ідентичними.

Для визначених відношень сервісів, можна сформулювати наступні теореми [9].

Теорема 1. Якщо S_i та S_j є виконуваними в $\mathcal{A}$ відносно $\mathcal{T}$, то $\mathcal{I} \Rightarrow_{S_i}^{\mathcal{T},\mathcal{A}} \mathcal{I}'$, $\mathcal{I} \Rightarrow_{S_j}^{\mathcal{T},\mathcal{A}} \mathcal{I}''$, у всіх моделях $\mathcal{I}$ ABox $\mathcal{A}$ відносно $\mathcal{T}$, якщо виконується $\mathcal{I}' \equiv \mathcal{I}''$, то $S_i \cong S_j$ у випадку ABox $\mathcal{A}$ відносно $\mathcal{T}$.

Теорема 2. Якщо S_i та S_j є виконуваними в $\mathcal{A}$ відносно $\mathcal{T}$, то $\mathcal{I} \Rightarrow_{S_i}^{\mathcal{T},\mathcal{A}} \mathcal{I}'$, $\mathcal{I} \Rightarrow_{S_j}^{\mathcal{T},\mathcal{A}} \mathcal{I}''$, у всіх моделях $\mathcal{I}$ ABox $\mathcal{A}$ відносно $\mathcal{T}$, якщо виконується $\mathcal{I}' \ll \mathcal{I}''$, то $S_i \succcurlyeq S_j$ у випадку ABox $\mathcal{A}$ відносно $\mathcal{T}$.

Відповідно до визначення відношення незалежності, можна визначити паралельні сервіси.

Паралельний сервіс [11] S – це сервіс, який досягається відношенням незалежності. $S = S_1 | S_2 | \cdots | S_k$, де $S_1 + S_2 + \cdots S_k$ еквівалентно $S_k + S_i + \cdots S_1$. Це означає, що сервіси-кандидати у сервісі S можуть виконуватися незалежно.

Сервіс $S = S_1 | S_2 | \cdots | S_k$ є виконуваним в $\mathcal{A}$ відносно $\mathcal{T}$ лише тоді, якщо кожний сервіс в S є виконуваним.

Згідно визначенню відношення передумови, можна вивести множину сервісів, що утворює послідовний сервіс.

Послідовний сервіс S [11] – це сервіс, що досягається відношенням передумови, а саме $S_1 \to S_2; \cdots; S_{k-1} \to S_k$.

Сервіс $S = S_1; S_2; \cdots; S_k$ є виконуваним в $\mathcal{A}$ відносно $\mathcal{T}$, лише тоді, якщо наступні умови вірні у всіх моделях $\mathcal{I}$ ABox $\mathcal{A}$ та TBox $\mathcal{T}$:

- $\mathcal{I} \vDash P_1$ та
- для всіх i з $1 \le i \le k$ та всіх інтерпретацій $\mathcal{I}'$ з $\mathcal{I} \Rightarrow_{S1;S2;\cdots;Si}^{\mathcal{T}} \mathcal{I}'$, маємо $\mathcal{I}' \vDash$ P(i+1) та
- $\mathcal{I} \Rightarrow_{S}^{\mathcal{T}} \mathcal{I}'$, $\mathcal{I}'$ не має конфліктів.

Композиція сервісів – це процес для виявлення кандидатів Web-сервісів, які можна скомбінувати у складний ланцюг, та для визначення порядку виконання для цих сервісів.

Формально, задачу композиції можна описати кортежем < $\mathcal{T}$, $\mathcal{A}$, G, S > [11], де:

- $\mathcal{T}$ описує словник прикладного домена;
- $\mathcal{A}$ містить твердження про іменовані індивіди в термінах цього словника,а також позначають вихідний стан світу;
- G – множина тверджень, яка представляє ціль, якої намагаються досягти;
- S – множина сервісів.

Для побудови композитного сервісу пропонується використовувати підходи та алгоритми на основі методів AI планування. Для цього необхідно визначити діаграму станів, щоб представити план та алгоритм для знаходження відповідних кандидатів сервісів, модель для представлення задачі декомпозиції та шляхи виконання плану. Діаграма станів будується зі станів та переходів. Діаграма переходів зі стану у стан позначається станами, умовами та операціями. Стани можуть бути простими або складними. Простий стан позначається одним компонентним сервісом – дією. Складний стан – це стан, який має графічну декомпозицію, яка може бути *OR* та *AND*-станами. *OR* – стани використовують механізм групування з метою модульності, тоді як *AND*-стан містить декілька областей, що призначені для одночасного виконання. Кожне твердження G можна транслювати в діаграми, які заміщують вузол задачі.

Таким чином, плани в діаграмі станів визначаються наступним чином.

Декомпозиція задачі композиції сервісів [11] – це послідовність декомпо-

зицій $[T_1, T_2, \ldots, T_n]$, така що T_1 – вихідна підзадача, T_n – кінцева підзадача, та для кожного стану T_i $(1 < i < n)$:

- T_i – є безпосереднім успішним результатом підзадач $[T_1, \ldots, T_{i-1}]$ та не є безпосереднім успішним результатом будь-яких інших станів в $[T_{i-1}, \ldots, T_n]$;
- $\neg\exists T_j \in [T_1, \ldots, T_{i-1}]$, де T_j та T_i належать OR-станам діаграми;
- якщо виконується вихідна задача однієї з конкурентних областей AND-стану, то виконуються всі інші гілки цього AND-стану.

Виходячи з цього, можна зробити наступні висновки щодо транслювання різних типів сервісів у стани діаграми: згідно визначенням паралельні сервіси можна транслювати в AND-стани; послідовні сервіси можна зв'язати один з одним; сервіси ідентичних відношень можна транслювати в OR-стани. Якщо діаграма містить OR-стани, вона має декілька шляхів з вихідного стану в кінцевий. Кожний шлях – це окремий план для завершення виконання складного сервісу. Структурування та підтримка шляху виконання відіграють ключову роль у підтримці ефективного планування.

Одним з ефективних інструментів представлення множини пов'язаних дій є направлений ациклічний граф (DAG). DAG [21] – це граф, ребрам якого присвоєне направлення, та який не містить циклів, тобто таких шляхів, що починаються та закінчуються в одній і тій самій вершині. DAG забезпечує топологічний пошук, який надає допустимий (загальний) порядок для виконання базових дій по одному. Діаграма станів перетворюється у план виконання, що представлений DAG $G = G(V, E)$, де $V = \{v_1, \ldots, v_n\}$ – множина сервісів та E – множина зважених направлених дуг, які ідентифікують залежності. Для кожної задачі T_i у діаграмі станів існує множина сервісів-кандидатів, які можуть досягати цільового стану T_i.

Формально, DAG шляху досягнення задачі визначається наступним чином.

Визначення 6 [11]. Задана діаграма станів декомпозиції задачі $[T_1, T_2, \ldots, T_n]$, DAG $G = G(V, E)$-представлення плану:

- DAG має щонайменше два вузли *Start* та *Finish*;
- $S_i = (S_{i1}, S_{i2}, \ldots, S_{it})$ – множина кандидатів сервісів для задачі T_i в ST та $E_i \supseteq T_i$;
- DAG містить один вузел для кожного сервісу $(S_1, S_2, \ldots, S_n)$;
- DAG містить ребро від сервісу S_i до дії S_j, лише тоді, якщо T_j – прямий послідовник T_i.

Додатково, можуть бути визначені операції об'єднання та перетину.

Визначення 7 (сформульовано на базі операцій об'єднання та перетину, що введені в [11]). Якщо задані два шляхи виконання, які представлені в DAG, $G_1 = G_1(V_1, E_1)$ та $G_2 = G_2(V_2, E_2)$, та кінцевий вузол G_1 є начальним вузлом G_2, тоді об'єднання G_1 та G_2 $(G_1 \cup G_2)$ – це також шлях виконання DAG. Нехай $G_3 = (V_3, E_3)$ – це $G_1 \cup G_2$, що створюється згідно наступним крокам:

1) всі вузли в G_1 та G_2 розглядаються як атомні вузли $V_3 = V_1 \cup V_2$;

2) якщо вузол $V_c \notin V_1 \cap V_2$, то DAG G_3 розширюється V_c та всіма ребрами, що відповідають V_c;

3) якщо вузол $V_c \in V_1 \cap V_2$, тоді порівнюються ребра E_{1c}, що відповідають V_c в G_1, з ребрами E_{2c}, що відповідають V_c в G_2, тоді обираємо оптимальні ребра, щоб розгорнути G_3.

Якщо задані два шляхи виконання, які представлені в DAG, $G_1 = G_1(V_1, E_1)$ та $G_2 = G_2(V_2, E_2)$, й кінцевий вузол G_1 збігається з кінцевим вузлом G_2, та $(V_1 \sqcap V_2) - (\{first, finish\}) \neq \emptyset$, тоді перетин G_1 та G_2 $(G_1 \cap G_2)$ – також є шляхом виконання DAG. Нехай $G_3 = (V_3, E_3)$ - $G_1 \cap G_2$ створюється згідно наступним крокам:

1) всі вузли в G_1 та G_2 розглядаються як атомні вузли, нехай $E_3 = E_1 - E_2$ та V_3 – множина, яка складається з вузлів, які примикають до кожного з ребер в E_3;

2) якщо вузол $V_c \in V_1 \cap V_2$, та $V_c \notin V_3$, тоді додамо V_c та пов'язані дуги, щоб розгорнути G_3.

Використовуючи ці дві операції, можна інтегрувати дрібно-деталізований план в крупно-деталізований або розділити задачу великого плану на декілька дрібно-деталізованих компонент.

Якщо говорити про реалізацію описаних методів планування, то такі техніки існують, наприклад, метод HTN планування [12]. Ключовим моментом цього глобального алгоритму є побудова бібліотеки планів, що представляються діаграмами станів.

Загальний процес складається з трьох кроків:

1) отримання запиту користувача;

2) планувальник шукає бібліотеку плану, щоб знайти модель діаграми станів та визначити підцілі;

3) обирається шлях, який може бути застосований згідно елементам контексту, та генерується шлях виконання DAG.

Однак, алгоритм HTN планування має ряд специфічних для нього обмежень [13]. Ці обмеження створюють проблеми щодо його застосування для задачі композиції Web-сервісів. Щоб їх подолати, в [13] пропонується формалізм планування HTN-DL, який комбінує HTN формалізм з представленням ДЛ. Його ключові відмінності відносно класичних HTN систем можна сформулювати у наступних категоріях.

Опис задач. Задачі описуються із використанням онтологій та співставленням задачі з операторами та методами, що зроблені на базі онтології задач, а саме: символи задач представляються як концепти та оператори ДЛ, а методи – як екземпляри. Співставлення задач частково скорочується до задачі пошуку екземпляра в ДЛ. Окрім цього, з задачами пов'язуються передумови та ефекти, так, що забезпечується більше інформації про задачу, яка використовується також для визначення співставлення сервісів.

Опис оператора/метода. Визначення оператора та метода використовують запити ДЛ для опису умов з передумов та ефектів. Ефекти сервісів, які традиційно не розглядаються в системах планування, представляються як екзистенціональні змінні в описах ефектів. Ефекти знань дій виражаються окремо так, що інформаційні сервіси можуть відрізнятися від сервісів, що змінюють світ.

Представлення станів. Стан світу описується як база знань ДЛ. Це дозволяє використовувати дуже виразну мову представлення знань для представлення інформації про світ. Традиційне в ДЛ, твердження відкритого світу адоптується у міркування.

Тобто ДЛ використовується як для опису дій, так й для опису станів як виразна мова опису знань. Стислий огляд формалізму, а також його формальний синтаксис та семантика наводиться у [13].

1. *http*://www.w3.org/2002/ws/
2. *Staab S., Studer R.* Handbook on Ontologies. Second edition.
3. *Baader Franz, Lutz Carsten, Miliˇci´c Maja, Sattler Ulrike, Wolter Frank.* A Description Logic Based Approach to Reasoning about Web Services. IN PROCEEDINGS OF THE WWW 2005 WORKSHOP ON WEB SERVICE SEMANTICS (WSS2005).
4. *Lifschitz V.* Frames in the space of situations. AIJ, 46:365–376, 1990.
5. *Sandewall E.* Features and Fluents. Oxford University Press, 1994.
6. *Winslett M.* Reasoning about action using a possible models approach. In Proc. of AAAI-88, pages 89–93, Saint Paul, MN, 1988.
7. *Reiter R.* Knowledge in Action. MIT Press, 2001.
8. *Ruben Lara.* Definition of semantics for web service discovery and composi tion. In Knowledge Web Deliverable D2.4.2, 2004.
9. *D2.4.6* A Theoretical Integration of Web Service Discovery and Composition. Roberti Pierluigi (ITC-IRST) Marco Pistore (University of Trento) with contributions from: Walter Binder (EPFL), Ion Constantinescu (EPFL) Axel Polleres (UIBK), Holger Lausen (UIBK), Paolo Traverso (ITC-

IRST), Michal Zaremba (NUIG). 2005. KWEB/2005/D2.4.6A/v1.0
10. *Semantic* Matching of Web Service Capabilities. Massimo Paolucci, Takahiro Kawamora, Terry R. Payne, Katya Sycara.
11. *Semantic* Web Services Composition Using AI planning of Description Logics. Lirong Qiu*" Fen Lin*" Changlin Wan*" Zhongzhi Shi* *Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China "Graduate School of the Chinese Academy of Sciences, 100039, Beijing, China {qiulr, linf, wancl, shizz}@ics.ict.ac.cn
12. *Kutluhan Erol, James Hendler, and Nau Dana S.* Htn planning: complexity and expressivity. In Proceedings of the twelfth national conference on Artificial intelligence (vol. 2), pages 1123–1128. American Association for Artificial Intelligence, 1994.
13. *Combining* Description Logic Reasoning with AI Planning for Composition of WEB Services. Evren Sirin, Doctor of Philosophy, 2006.
14. *http://*www.dh.cs.fau.de/IMMD8/Lectures/KRR/08a-DL_KB-4.pdf
15. *https://*www.ics.forth.gr/tech-reports/2010/2010.TR409_Automated_Web_Service_Composition.pdf
16. *https://*ru.wikipedia.org
17. *Web* Service composition: Semantic Links based approach. Freddy L´ecu´, Doctor of Philosophy, 2008.
18. *A Conceptual* and Formal Framework for Semantic Web Services (v1). Holger Lausen, Francisco Martin-Recuerda, Jos de-Bruijn, and Michael Stollberg (University of Innsbruck)
19. *A Proposal* for Describing Services with DLs. Carsten Lutz and Ulrike Sattler {lutz, sattler}@cs.rwth-aachen.de
20. *Semantic* Web Service Composition Based on a Closed World Assumption. Freddy L´ecu´ e, Alain L´eger, France Telecom R&D, France 4 rue du clos courtel, F-35512 Cesson S´evign´ e {(freddy.lecue, alain.leger)@orange-ft.com}, ´Ecole Nationale Sup´erieure des Mines de St-Etienne, France 158, cours Fauriel, F-42023 Saint-´Etienne, http://ieeexplore.ieee.org, 2006
21. *http://*life-prog.ru/view_zam2.php?id=204&cat=5&page=13
22. *Integrating* Description Logics and Action Formalisms for Reasoning about Web-services. Franz Baader, Carsten Lutz, Maja Milieie, Ulrike Sattler, Frank Wolter. LTCS-Report 05-02
23. *http://*www.roseindia.net/webservices/Web-Services-technology.shtml
24. *http://*www.informit.com/articles/article.aspx?p=336265
25. *Formal* Description of Web Services for Expressive Matchmaking. Dipl.-Inform. Sudhir Agarwal, 2007 Karlsruhe

Одержано 11.09.2014

***Про автора*:**

Захарова Ольга Вікторівна,
кандидат технічних наук,
старший науковий співробітник.

Місце роботи автора:

Інститут програмних систем
НАН України,
Проспект Академіка Глушкова, 40.
Тел.: 526 51 39.
E-mail: ozakharova68@gmail.com.