УДК 004.94

О. Захарова

МЕТОДИКА ЗАСТОСУВАННЯ АПАРАТА ДЕСКРИПТИВНИХ ЛОГІК У ПРОЦЕСІ ПОБУДОВИ КОМПОЗИТНОГО СЕРВІСУ НА ФУНКЦІОНАЛЬНОМУ РІВНІ

Автоматизована композиція – найскладніша задача веб-сервісів. Вона має дві цілі: з одного боку – задовільнити складні вимоги клієнта, а з іншого – зменшити складність розробки веб-сервісу, щоб задовільнити складну прикладну систему. Створений у роботі теоретичний апарат є основою для розробки алгоритмів автоматизованого вирішення задачі композиції семантичних веб-сервісів. Запропоновані підходи базуються на використанні апарата дескриптивної логіки, що завдяки своїм механізмам міркувань та можливостям логічного виводу та надання описам семантичного змісту, є ефективним та потужним інструментом.

Ключові слова: семантичний Веб-сервіс, дескриптивна логіка, задача композиції, онтологія сервісу, онтологія композиції, IOPE-модель.

Вступ

Композиція Веб-сервісів являє собою комбінування та координування множини сервісів з метою досягнення функціональності, що не може бути реалізована існуючими сервісами. Тобто, це можливість забезпечити нову функціональність, отриману з комбінації різних сервісів, що надаються різними провайдерами. Для отримання такого композитного сервісу необхідно вміти знаходити найбільш відповідні Веб-сервіси для композиції та мати автоматичний інструмент для їх композиції. Зрозуміло, що бажано, щоб створення такого сервісу здійснювалося автоматизованим чином. Ефективність вирішення цих та інших супутніх задач та вибору алгоритмів для їх вирішення, перш за все, залежить від обраної моделі формалізації сервісу.

В даній роботі будемо розглядати семантизовані сервіси, що представлені функціональною IOPE моделлю, та для формального представлення яких використовується дескриптивна логіка (ДЛ). Задача композиції породжує багато супутніх задач на всіх етапах життєвого циклу сервісів: від їх формалізації, формалізації пошукового запиту, виявлення та інші. Підходи до формалізації семантичних веб-сервісів та пошукового запиту засобами дескриптивної логіки та виявлення потрібних веб-сервісів шляхом співставлення сервісів та пошукового запиту бути детально розглянуті в [1]. Предметом розгляду цієї статті є побудова композитного сервісу на базі відібраної множини існуючих сервісів, що представлені IOPE моделлю. При цьому використовуються підходи встановлення відповідності за виходами – входами та ефектами – передумови сервісів, що композуються.

Моделі представлення сервісу на функціональному рівні

Залежно від повноти опису функціональних можливостей сервісу розрізняють різні формальні моделі представлення. Так, в IO моделі сервіс описується лише наборами вхідних та вихідних параметрів. PE модель має на увазі представлення сервісу його передумовами та ефектами. У загальному випадку, опис функціональних можливостей сервісу визначається його входами, виходами, передумовами та ефектами – IOPE моделлю. Іноді IOPE модель розширюється постумовами, або/та об'єктами ефектів. Наочно, IO (входи, виходи) надають синтаксичну інформацію про веб-сервіси, тоді як PE (передумови та ефекти) – відображають їх семантику. Методи, які використовуються у співставленні виходів-входів відрізняються від тих, що використ-

товуються для передумов та ефектів. Та семантики, які відображаються входами-виходами й передумовами та ефектами також різні. На сьогодні досягнуто значних результатів у співставленні веб-сервісів за входами-виходами, але відсутній ефективний підхід щодо співставлення передумов та ефектів. У роботі [2] наводиться алгоритм PE співставлення, на основі формалізма ДЛ SHOIN+(D).

В даній роботі розглядаються IO та IOPE моделі, що для формального опису входів, виходів, передумов та ефектів використовують апарат дескриптивної логіки (ДЛ). Безперечна доцільність використання апарата ДЛ для вирішення багатьох задач Веб-сервісів обумовлена тим, що системи ДЛ забезпечують користувачів різними механізмами виводу, які виводять неявні знання з тих, що явно представлені, та більше того, на сьогодні існує вже чимало реалізованих механізмів міркувань (резонерів) ДЛ, що готові до використання. Але, варто пам'ятати, що окремою складною задачею залишається вибір такої ДЛ, що є компромісним рішенням між її виразністю та розв'язуваністю.

При визначені формального опису веб-сервісу на функціональному рівні засобами ДЛ описується онтологія домена. Зрозуміло, що онтологія домена – специфічна для кожної предметної області, але при вирішенні задач веб-сервісів вона має бути єдина (або інтегрованою) для всіх сервісів, що прийматимуть у цьому участь. Анотації входів, виходів, передумов та ефектів веб-сервісів є аксіомами визначення або аксіомами включення ДЛ з посиланнями на онтологію домена.

Але, перш за все, для вирішення задач веб-сервісів доцільно мати онтологію сервісу вищого рівня, що надаватиме загальне визначення веб-сервісів. На цей опис будуть спиратися описи конкретних веб-сервісів, що використовуватимуться при вирішенні задач. Онтологія сервісу має відображати визначення його моделі, тому її TBox залежить від обраної функціональної моделі. Для IOPE моделі онтологію сервісу ***ServiceOntology*** можна визначити наступним чином.

TBox

Service, InputParameter, OutputParameter,

PreCondition, PostCondition, Parameter,

Name, Value, Type

Service $\sqsubseteq$ *has.InputParameter;* /* *I* –

складова сервісу

Service $\sqsubseteq$ *has.OutputParameter* /* *O* –

складова сервісу

Service $\sqsubseteq$ *has.PreCondition;* /* *P* –

складова сервісу

PreCondition $\sqsubseteq$ *Condition*

PostCondition $\sqsubseteq$ *Condition;*

Condition $\sqsubseteq$ *hasValue.Boolean*

Service $\sqsubseteq$ *has.PostCondition;* /* *E* –

складова сервісу

InputParameter $\sqsubseteq$ *Parameter*

OutputParameter $\sqsubseteq$ *Parameter;*

Parameter $\sqsubseteq$ *has.Name*

Parameter $\sqsubseteq$ *has.Value;*

Name $\sqsubseteq$ *String;*

Value $\sqsubseteq$ *has.Type*

Визначення онтологій конкретних сервісів та онтології домена буде розглянуто далі на конкретному прикладі застосування методології, що пропонується в роботі для вирішення задачі композиції.

Огляд існуючих підходів до композиції веб-сервісів

Мета задачі композиції – побудова складного сервісу, який реалізує поставлену бізнес-задачу. Задача композиції виражається запитом на композицію, що являє собою опис сервісу, який потрібно отримати. Результуючий композитний сервіс має відповідати запиту на композицію, або іншими словами, «покривати» цей запит.

Одним з підходів для формування такого сервісу є використання задачі знаходження найкращого покриття запиту [3, 4], що з технічної точки зору належить

до загальної структури для рерайтингу (перезапису) [6, 7], використовуючи термінологію з [5].

Знаходження найкращого покриття

За умови представлення сервісів засобами ДЛ, кожний елемент визначення сервісу та запиту є концептом (класом), визначенням якого є його опис.

Семантики описів концептів визначаються в термінах інтерпретації $\mathcal{J}=$ $=(\Delta^{\mathcal{J}},.^{\mathcal{J}})$, що складається з непустої множини $\Delta^{\mathcal{J}}$ – інтерпретації домена та функції інтерпретації $.^{\mathcal{J}}$, яка зв'язує з кожним іменем концепта $P \in C$ підмножину $P^{\mathcal{J}}$ of $\Delta^{\mathcal{J}}$ та з кожним іменем ролі $R \in \boldsymbol{\mathcal{R}}$ бінарне відношення $R^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$.

На основі цих семантик визначаються subsumption (включення), еквівалентність та поняття найменшого загального включення [3], що є основою для отримання найкращого покриття.

Нехай C, C1,...,Cn та D – описи концептів:

– C включається в (subsumed by) D ($C \sqsubseteq D$), якщо $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$ для всіх інтерпретацій $\mathcal{J}$.

– C еквівалентно D ($C \equiv D$), якщо $C^{\mathcal{J}} = D^{\mathcal{J}}$ для всіх інтерпретацій $\mathcal{J}$.

– D – найменше загальне включення концептів $C_1,...,C_n$ ($D = lcs(C_1,...,C_n)$), якщо:

- $C_i \sqsubseteq D$ для всіх $1 \le i \le n$, та
- D найменший опис концепта, що має таку властивість, тобто, якщо D' – опис концепта, що задовільняє $C_i \sqsubseteq D$ для всіх $1 \le i \le n$, тоді $D \sqsubseteq D'$ [3].

Таким чином, найменше загальне включення множини концептів відповідає найбільш специфічному опису, що містить всі концепти з множини. Більш детально питання побудови найменшого загального включення висвітлюється в [4].

Процес виявлення сервісів можна розглядати як процес перезапису (rewriting), в якому запит Q перезаписується найближчим до нього описом E, який виражається кон'юнкцією Веб-сервісів заданої онтології $\boldsymbol{\mathcal{T}}$, на базі якої визначаються сервіси та запит. Такий опис і є найкращим покриттям запиту.

Представлення сервісів та запиту як концептів ДЛ дозволяє також використовувати операцію різності на описах сервісів, що забезпечує гнучкий механізм співставлення сервісів та запиту або входів та виходів сервісів. У роботі [8] різність задається як найменше загальне включення двох концептів.

Зрозуміло, що є малоймовірним знайти покриття яке повністю відповідає запиту, тобто не містить зайвої інформації або/та не має відсутньої інформації. Тому, найкраще покриття – це покриття, яке, по-перше, має найменшу решту, по-друге – найменшу частину відсутньої інформації. Задача найкращого покриття – це задача обчислення всіх найкращих покриттів запиту $\boldsymbol{Q}$, використовуючи термінологію задану $\boldsymbol{\mathcal{T}}$. Мінімізація залишку та нестачі інформації і є критерієм оптимальності при вирішенні даної задачі.

Слід зазначити, що знаходження покриття з нуля для кожного запиту це досить трудомістка задача. Окрім цього, сервіси, що утворюють це покриття необхідно впорядкувати в ланцюжки, щоб вони утворили ефективну композицію. Тому, доцільно було б визначати та зберігати залежності між серсвісами у реєстрі. Визначення всіх залежностей між усіма сервісами утворює, так званий, ***граф залежностей*** [9]. Будь-який ланцюжок, що є його підмножиною, утворює складний сервіс. Таким чином, для побудови композитного сервісу, що реалізовуватиме поставлену бізнес-задачу, треба знайти такий ланцюжок в графі залежностей, що на вході прийматиме вхідні параметри та передумови, визначені в запиті на композицію, а за досягненням кінцевого вузла забезпечувати необхідні виходи та ефекти. Найчастіше пошук такого ланцюжка здійснюється саме з кінця, тобто з сервіс-вузла, що забезпечує виходи запита. У такому випадку, кажуть, що сервіс будують знизу вгору. Тобто, починаючи з кінцевого атомного сервісу, підіймаються по ланцюжку вгору аж доки не будуть досягнуті не-

обхідні входи запиту на композицію. Зрозуміло, що для побудови графа залежностей (та композиції сервісів) необхідно враховувати та вміти визначати зв'язки між різними Web-сервісами, що використовуються в композиції. Ці відношення можна класифікувати.

Класифікація залежностей між веб-сервісами

Визначення 1 [10]. Нехай $\mathcal{T}$ – ациклічний Tbox, $\mathcal{A}$ - 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 – вільно незалежний від іншого та порядок їх виконання не впливає на результат.

- *Ідентичність:* $S_i = S_j$ – два сервіси забезпечують одну й ту саму послугу, але вони мають деякі різні атрибути.
- *Умовна ідентичність*: $S_i \cong \cong S_j - S_i$ 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 сервісами. Для створення цього відношення, має бути виключена частина, що є перекриттям.
- *Передумова:* $S_i \rightarrow S_j$ – один сервіс має завершитися до початку другого. Сервіс S_i має завершитися до того, як почнеться сервіс S_j.

Для визначених відношень сервісів, можна сформулювати наступні теореми [11].

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 [10] S – це сервіс, який досягається відношенням незалежності. $S = S_1 | S_2 | \cdots | S_k$, де $S_1 + S_2 + ... + S_k$ 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 [10] – це сервіс, що досягається відношенням передумови, а саме $S_1 \rightarrow S_2 ; \cdots ; S_{k-1} \rightarrow S_k$.

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

- $\mathcal{I} \vDash P_1$ та
- для всіх *i* з $1 \le i < k$ 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.

Підходи до побудови композитного сервісу, що засновані на графах

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

Формально, задачу композиції можна описати кортежем < $\mathcal{T}$, $\mathcal{A}$, G, S > [10], де:

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

Можливі різні підходи для побудови композитного сервісу, але вони мають спільні складові, а саме: використання запиту, як цілі композиції, фільтрація сервісів репозиторію для виявлення відповідних сервісів, співставлення сервісів за входами/виходами, передумовами та ефектами для побудови результуючого ланцюжка. Підходи, що використовують направлені ациклічні графи для представлення композитного сервісу, відносяться до групи підходів з доведеною ефективністю. Причому алгоритм побудови можливий у двох напрямках: 1) виходячи від декомпозиції цілі, тобто аналогічно до підходів AI-планування, 2) пошук всіх можливих графів з сервісів репозиторію, що відповідатимуть запиту.

Щоб для побудови композитного сервісу застосувати підходи та алгоритми на основі методів AI планування, 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 із станів та переходів. Діаграма переходів із стану в 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 вузел задачі.

Таким чином, плани в діаграмі станів визначаються наступним чином.

Декомпозиція задачі композиції сервісів [10] – це послідовність декомпозицій $[T_1, T_2, ..., T_n]$, така що T_1 – вихідна підзадача, T_n – кінцева підзадача, та для кожного стану T_i $(1 < i < n)$:

- T_i – безпосередній успішний результат підзадач $[T_1, ..., T_{i-1}]$ та не є безпосереднім успішним результатом будь-яких інших станів в $[T_{i-1}, ..., T_n]$.
- $\neg\exists\, T_j \in [T_1, ..., T_{i-1}]$, 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 сервіса. Структурування та підтримка шляху виконання відіграють ключову роль у підтримці ефективного планування.

Ациклічний направлений граф (DAG), у даному випадку, є ефективним інструментом представлення множини пов'язаних дій. DAG [12] – 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 $V = \{v_1, ..., v_n\}$ – 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 [10]. Задана діаграма станів декомпозиції задачі $[T_1, T_2, ..., T_n]$, DAG $G = G(V, E)$ -представлення плану:

- DAG має щонайменше два вузли *Start* та *Finish*.
- $S_i = (S_{i1}, S_{i2}, ..., S_{it})$ – множина кандидатів сервісів для задачі T_i в ST та $E_i \supseteq T_i$.
- DAG містить один вузел для кожного сервіса $(S_1, S_2, ..., S_n)$.
- DAG містить ребро від сервіса 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 (сформульовано на базі операцій об'єднання та перетину, що введені в [10]). 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 вузел 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 вузел $V_c \notin V_1 \cap V_2$, то DAG G_3 розширюється V_c та всіма ребрами, що відповідають V_c;

3) якщо вузел $V_c \in V_1 \cap V_2$, 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 вузел G_1 співпадає 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 вузел $V_c \in V_1 \cap V_2$, 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.

Зв'язки між веб-сервісами можна визначати й за ходом виконання композиції (без попередньої декомпозиції цілі). Наприклад, для визначення залежностей сервісів можна використовувати графову модель.

Використання графової моделі для автоматичної композиції сервісів. В даному випадку, поведінка наявних веб-сервісів представляється в термінах їх інформації входів-виходів та семантичної інформації про веб-дані. Граф залежностей [9] є набором вершин або «вузлів» та ребер, що з'єднують пари вершин. Граф може бути неорієнтовний, це означає, що немає ніякої різниці між двома вершинами, пов'язаних з кожним ребром, або його ребра можуть бути спрямовані від однієї вершини до іншої. Зважений граф являє со-

бою граф, де кожне ребро має вагу (деяке дійсне число), що з ним пов'язана. В пошуці композитного сервісу використовується граф залежностей, який виконує даний запит.

Більшість заснованих на графах методів композиції будують графи залежностей веб-сервісів під час виконання. Вони використовують алгоритм пошуку для обходу графів залежностей для того, щоб композувати сервіси. Основна відмінність між такими методами полягає в тому, як вони шукають граф залежностей: пряма побудова ланцюжка, зворотна побудова ланцюжка, двонаправлені алгоритми пошуку тощо.

В роботі [13] визначається задача *узагальненої композиції*, як підхід, що полягає в автоматичному знаходженні у репозиторії орієнтованого ациклічного графа сервісів, які можуть бути компоновані для отримання сервісу, заданого запитом. Формально, необхідно в репозиторії R знайти орієнтований ациклічний граф $G = (V, E)$ сервісів відповідно до запиту на композицію Q, де V – набір вершин і E – набір ребер графа. Сервіс у IOPE моделі являє собою кортеж з 4 елементів $S = (CI, I, O, CO)$, де CI – список передумов, I – список входів, O – список виходів і CO – список післяумов (або ефектів). Перед- та післяумови являють собою заземлені логічні предикати (ground logical predicates). Аналогічно, запит на сервіс визначається як $Q = (CI', I', O', CO')$, де CI' – список передумов, I' – список входів, O' – список виходів, CO' – список пост-умов. Кожен вузол у графі є або сервісом, що бере участь в композиції, або пост-умовою безпосереднього попереднього сервісу в графі, результат якого може бути визначений тільки після виконання сервісу. Кожне вихідне ребро вузла (сервіс) представляє виходи і післяумови, вироблені сервісом. Кожне вхідне ребро вузла представляє входи і передумови сервісу. На вузлах графа мають виконуватись умови:

1. $\forall i \quad S_i \in V$, де S_i має рівно одне вхідне ребро, яке представляє входи запиту і передумови,

$$I' \sqsupseteq \bigcup_i I_i,\ CI' \Rightarrow \wedge_i CI_i.$$

2. $\forall i \quad S_i \in V$, де S_i має рівно одне вихідне ребро, яке представляє виходи запиту і пост-умови,

$$O' \sqsubseteq \bigcup_i O_i,\ CO' \Leftarrow \wedge_i CO_i.$$

3. $\forall i \quad S_i \in V$, де S_i є сервісом і має, щонайменше, одне вхідне ребро. Нехай $S_{i1}, S_{i2}, ..., S_{im}$ будуть вузли такі, що існує спрямоване ребро від кожного з цих вузлів до S_i. Тоді

$$I_i \sqsubseteq \bigcup_k \mathcal{O}_{ik} \cup I',\ CI_i \Leftarrow (\mathcal{CO}_{i1} \wedge \mathcal{CO}_{i2} ... \wedge \mathcal{CO}_{im} \wedge CI').$$

4. $\forall i \quad S_i \in V$, де S_i представляє умову, яка обчислюється під час виконання, і має рівно одне вхідне ребро, нехай S_j буде його безпосереднім вузлом-попередником, таким, що існує спрямоване ребро від S_j до S_i. У такому разі, для входів та передумов у вузлі S_i виконується: $I_i = O_j \cup I'$, $CI_i = CO_j$. Вихідні ребра з S_i представляють виходи, які співпадають з входами I_i, та пост-умови, що є результатом умови, обчисленої під час виконання, які є передумовами для S_i.

У даному випадку, $\sqsubseteq$ є відношенням включення, а $\Rightarrow$ - відношенням імплікації. Іншими словами, сервіс на будь-якому етапі в композиції може потенційно мати як свої входи всі виходи від своїх попередників, а також входи запиту. Сервіси на першому етапі композиції можуть використовувати тільки входи запиту. Об'єднання виходів, вироблених сервісами, на останній стадії композиції, має містити всі виходи, які визначені у запиті. Постумови сервісів на будь-якому етапі композиції мають означати передумови сервісів на наступному етапі. Не завжди при виконанні можна визначити, чи має місце імплікація між постумовами та передумовами наступного сервісу. В такому випадку, в графі створюється умовний вузол. Вихідні ребра умовного вузла представляють мож-

ливі умови, які будуть обчислюватися під час виконання. Залежно від умови, що є істиною, виконуються відповідні сервіси. Тобто, якщо підсервіс S_1 композований з під-сервісом S_2, то пост-умови CO_1 повинні імплікувати передумови CI_2 з S_2. Під час виконання обчислюються наступні умови:

if ($CO_1 \Rightarrow CI_2$) then execute S_1;

else if ($CO_1 \Rightarrow \neg CI_2$) then no-op;

else if (CI_2) then execute S_1.

Зазначені формалізми дозволяють визначити ітераційний алгоритм композиції [13], де пошук та фільтрація відповідних сервісів відбувається на основі множини вхідних параметрів та передумов, що формується ітераційно на кожному кроці.

Композитний сервіс є графом. Щоб його отримати, перш за все необхідно відфільтрувати сервіси репозиторію, що не є корисними для композиції на декількох етапах (рис. 1).

Якщо процедура композиції здійснюється згори донизу, то вона починається з вхідних параметрів запиту. Алгоритм знаходить всі сервіси з репозиторію, які як вхідні потребують підмножину вхідних параметрів запиту.

На рис. 2 CI, I є передумови та вхідні параметри, визначені в запиті. S_1 і S_2 – сервіси, знайдені після першого кроку – фільтрації за вхідними параметрами запиту. O_1 є об'єднання всіх виходів, вироблених сервісами на першому етапі. На наступному етапі, доступні входи – це вхідні параметри запиту та всі виходи, що отримані на попередньому етапі, тобто $I_2 = O_1 \cup I$. Саме I_2 й використовується для пошуку сервісів на наступному кроці, тобто всі ті сервіси, які потребують підмножину I_2 як вхідні параметри. Щоб алгоритм не зациклився, отримуються лише ті сервіси, які вимагають щонайменше один параметр з виходів, отриманих на попередньому етапі. Така фільтрація триває, поки не будуть вироблені всі вихідні параметри запиту. Після цього здійснюється ще один прохід у зворотному напрямку, щоб видалити надлишкові сервіси, які прямо або побічно не сприяють вихідним параметрам запиту. В даному випадку, треба починати з вихідних параметрів та рухатися у зворотному напрямку.

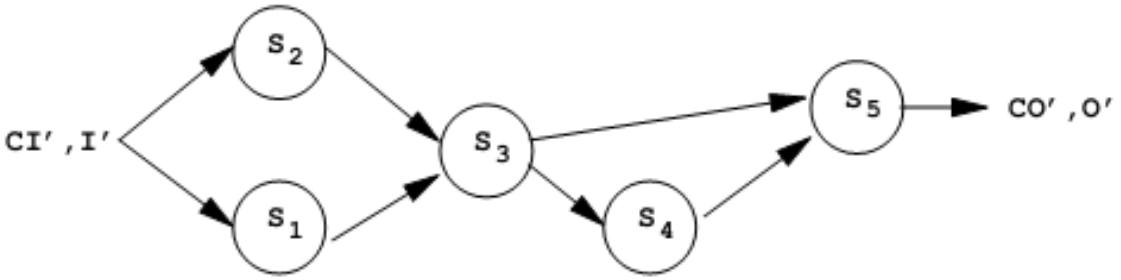

Рис. 1. Композитний сервіс, побудований з п'яти сервісів, як орієнтований ациклічний граф

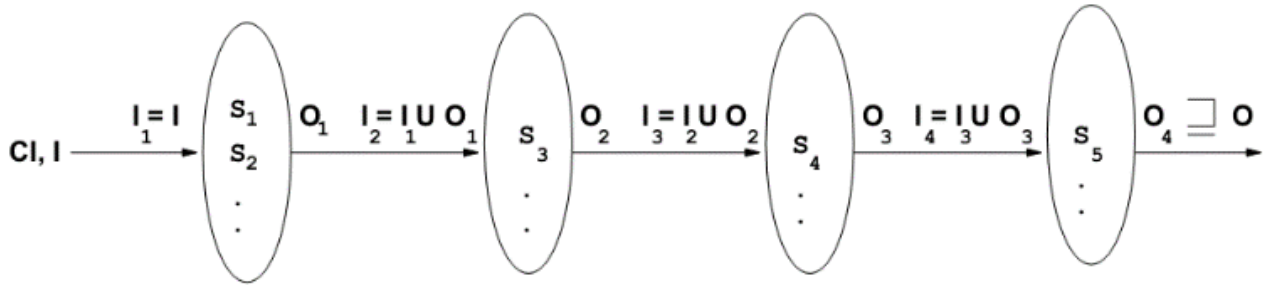

Рис. 2. Алгоритм узагальненої композиції

Кожен композиційний сервіс, породжений наведеним алгоритмом, є дійсно сервісом, що відповідає всім вимогам запиту і, отже, є правильним. Окрім цього, для заданого запиту алгоритм генерує всі можливі композитні сервіси, тобто, є повним.

Але, очевидно, що на кожному кроці може бути декілька можливих шляхів у графі, та, як слідство, декілька результуючих композиційних сервісів. Тому, неодмінно виникає питання знаходження найкоротшого шляху в графі. Визначення ваги для всіх ребер графа дозволило б оптимізувати композицію шляхом обчислення найкоротших шляхів.

Одним з підходів визначення найкоротшого шляху у зваженому ациклічному графі є використання матриці найкоротшого попередника. В даному випадку всі результати виконання алгоритму пошуку зберігаються в матриці, а потім ця матриця використовується для реконструкції найкоротшого шляху, який відповідає оптимальній автоматизованій композиції сервісів. Граф та матриця мають оновлюватися кожного разу, коли до репозиторію додаються нові сервіси або існуючі сервіси змінюються, або видаляються з репозиторію. Це дозволяє уникнути побудови графа і застосування алгоритму пошуку графа кожного разу, коли здійснюється запит на композицію сервісів. Під час виконання композиції і при обробці кожного запиту, визначаються лише початкові й кінцеві веб-сервіси в графі. А потім з матриці витягується відповідний найкоротший шлях між початковою та кінцевою вершиною. Ключовим моментом такого підходу є вага, так як вона впливає на вибір шляхів композиції. Існують різні підходи до визначення ваги. У роботі [14] пропонується алгоритм зважування, при якому в відповідність всім ребрам у графі ставиться вага, розрахована з використанням значення S семантичної подібності між вхідними і вихідними параметрами, і функції f(QoS) нефункціональних властивостей сервісів, як, наприклад, вартість, час виконання, надійність, доступність і т. д.

Так, наприклад, з урахуванням трьох параметрів, а саме: вартості, часу і наявності, f(QoS) для ребра i можна обчислити наступним чином:

$$f(QOS(V_i)) = (\alpha * cost) + (\beta * execution\ time) + (\mu * availability), \quad (1)$$

де α, β і μ – це відносні фактори, які можуть бути визначені системним адміністратором. Тоді, вагу W_{ij} даного ребра $V_i \to V_j$ можна обчислити наступним чином:

$$W_{ij} = f(QoS(V_i)) + S_{ij}, \quad (2)$$

де S_{ij} – значення семантичної подібності між вхідними параметрами V_j і вихідними параметрами V_i, та приймає одно з чотирьох значень (Exact, Plug-in, Subsumes, Fail).

Початкові та цільові вершини в графі для кожного запиту визначаються також, на основі семантичних подібностей, але між вхідними і вихідними параметрами запиту та вершинами графу.

Одним з найпростіших у реалізації алгоритмів обчислення найкоротших шляхів [15] є алгоритм Флойда – Воршалла [16]. Його часова складність займає $\theta(N^3)$, де N – кількість вершин. Одне виконання алгоритму забезпечить довжини (сумарні ваги) найкоротших шляхів між усіма парами вершин, хоча це не повне деталі про самі шляхи. Тому, у [13] автори описують зміни до алгоритму, які дозволили створити метод для реконструкції фактичного найкоротшого шляху між будь-якими двома кінцевими вершинами. Реконструкція шляху в цьому методі проходить в $\theta(N)$ і, таким чином, не впливає на складність алгоритму.

Методичні підходи до побудови композитного сервісу

Проведені дослідження дозволили визначити методичні підходи до побудови композитного сервісу, на основі функціональної IOPE моделі сервісу.

1. Запит – абстрактний опис сервісу, що реалізує необхідну функціональність. Таким чином, композитними характеристиками результуючого сервісу мають бути саме параметри запиту, або множина параметрів, що максимально наближена до вимог запиту.

2. Методика базується на використанні ДЛ як для представлення предметної області задачі (онтологія домена), так і для вирішення безпосередньо задач веб-сервісів, шляхом представлення сервісів та запиту за допомогою онтологій, де в якості мови опису онтологій обрано дескриптивну логіку.

3. Семантичні описи веб-сервісу являють собою твердження або аксіоми ДЛ. Задачі, що виникають при вирішенні задач веб-сервісів, зводяться до задач виконуваності ДЛ.

4. Кожен сервіс представляється ДЛ-онтологією на основі загальної онтології сервісу верхнього рівня. Кожна онтологія конкретного сервісу базується на базі знань інтегрованої ДЛ-онтології домену.

5. Враховуючі, що опис профілей семантичних веб-сервісів, що зберігаються в реєстрі, є WSDL документ, який доповнений семантичними анотаціями, тобто – звичайний XML, можливий автоматичний розбір цього XML-визначення та побудова онтології конкретного сервісу на основі онтології сервісу верхнього рівня, що визначена вище.

6. Алгоритм співставлення на основі ДЛ, що представлений в [1] може бути застосований для співставлення сервісів при композиції.

7. Існуючі резонери ДЛ дозволять здійснювати автоматичне співставлення запиту та сервісів, сервісів один з одним за входами виходами. Це стандартні задачі виводу ДЛ. Відношення між концептами та індивідами, що представлятимуть вхідну та вихідну інформацію запиту, визначають ступінь відповідності сервісів, що співставляються та ранжувати їх за цим показником.

8. В ході вирішення задач виявлення та композиції сервісів ймовірно виникає необхідність розширення онтології домену, уточнення онтологій сервісів чи описів сервісів. Резонери ДЛ дозволять перевірити отриманий опис на відсутність протиріч – стандартна задача розв'язуваності ДЛ.

9. Відповідність за виходами є пріорітетнішою ніж відповідність за входами. Тому, пошук сервісів та побудову ланцюжка доцільніше все ж таки проводити знизу-вгору – від виходів запиту.

10. Ітераційний покроковий алгоритм побудови ланцюжка:

a) пошук у реєстрі всіх сервісів, що відповідають вихідним параметрам та ефектам запиту, та ранжування їх за ступенем відповідності. Відповідно до критерія виявлення, найбільшу ступінь відповідності (*Exact*) мають сервіси, виходи та ефекти яких еквівалентні виходам та ефектам запиту. Наступну ступінь відповідності мають сервіси, виходи та ефекти яких включають (subsumes) виходи та ефекти запиту. Але розглядаються також сервіси, вихідні характеристики яких включають хоча б один вихід запиту (subsumed);

b) якщо знайдено сервіс з Exact або Subsumes відповідністю за виходами, то це єдиний кінцевий сервіс ланцюжка;

c) якщо існує декілька сервісів з Exact або Subsumes відповідністю за виходами, аналізуються їх вхідні характеристики (вхідні параметри та передумови), та обирається сервіс, найближчий до запиту за входами;

d) якщо існують сервіси тільки з Subsumed відповідністю виходів до запиту, обираємо сервіс, що має найбільший перетин за виходами із запитом;

e) виходами результуючого композитного сервісу буде об'єднання без повторень всіх виходів сервісів, що прийматимуть участь у ланцюжку;

f) сервіси кандидати-попередники шукаються за критерієм: їх виходи співставляються з множиною, що є об'єднанням виходів запиту та входів наступного сервісу. Аналогічно, постумови (ефекти) сервісу, що шукається, мають

бути пов'язані імплікацією з передумовами наступного сервісу або постумовами запиту. Сервіси ранжуються за ступенем відповідності. Після чого аналізуються входи відібраних сервісів, які перевіряються на відповідність вхідним параметрам запиту, а передумови повинні бути імплікацією передумов запиту. Якщо це не виконується, але є відповідність за виходами, сервіси вбудовуються у ланцюжок, але продовжується ітеративний процес пошуку сервісів – попередників, доки не будуть максимально задоволені всі вимоги запиту.

11. Підхід, що пропонується, полягає у застосуванні апарату ДЛ для реалізації описаної методики. При цьому, сама задача композиції описується також за допомогою апарата ДЛ. При такій постановці, задача відповідності двох сервісів зводиться до вирішення стандартної задачі розв'язуваності ДЛ, для чого може бути використаний будь-який з існуючих резонерів. Продемонструємо підхід на простому прикладі.

Приклад побудови композитного сервісу

Як приклад розглянемо просту задачу, а саме – задачу *побудови композитного сервісу для вирішення задачі знаходження S площі трикутника, що заданий трьома векторами* $\bar{a}$, $\bar{b}$, $\bar{c}$.

Так як поставлена задача є геометричною, то онтологія домену спирається на онтологію геометрії.

TBox онтології домену *POGeometry*

$Vertex \sqsubseteq has.XCoordinate$

$Vertex \sqsubseteq has.YCoordinate$

$XCoordinate \sqsubseteq Coordinate$

$YCoordinate \sqsubseteq Coordinate$

$Coordinate \sqsubseteq hasValue.NUMBER$

$Verterx \sqsubseteq has.VerterxLength$

$Verterx \sqsubseteq has.VerterxAngle$

$VerterxLength \sqsubseteq hasType.NUMBER$

$VerterxAngle \sqsubseteq hasType.NUMBER$

$Height \sqsubseteq hasType.NUMBER$

$LenthAB \sqsubseteq hasType.NUMBER$

...

$Triangle \sqsubseteq GeometricFigure$

$Triangle \sqsubseteq =3has.Vertex$

$Triangle \sqsubseteq =3has.Vector$

$Triangle \sqsubseteq =3has.Height$

$Square \sqsubseteq hasType.NUMBER$

$Triangle \sqsubseteq has.Square$

Онтології запиту та сервісів спиратимуться на онтологію сервісу вищого сервісу *ServiceOntology* та онтологію домена *POGeometry*.

Онтологію запиту можна представити наступним чином:

<u>*TBox*</u>

$Q \sqsubseteq Service$ /* зв'язок з ***ServiceOntology***

$Q \sqsubseteq =3has.InputParameter$

$Q \sqsubseteq =1has.OutputParameter$

$I_Q \sqsubseteq InputParameter$

$O_Q \sqsubseteq OutputParameter$

<u>*ABox*</u>

$a : I; b : I; c : I ; S: O$

$a : Verterx, b : Verterx, c : Verterx$ /* зв'язок з онтологією ***POGeometry***

$S : Square$

Припустимо, що в реєстрі нема сервісу, що задовільняє вимогам запиту, але ми знайшли сервіс S_4 з вхідними параметрами: a – *NUMBER*, h – *NUMBER*, та вихідним параметром S – *NUMBER*. Семантичні анотації сервісу вказують, що a – довжина однієї з сторін трикутника, h – його висота, що опущена на цю сторону, S – площа цього трикутника.

А саме, онтологія цього сервісу *S4Ontology* має наступний TBox:

TBox

$S_4 \sqsubseteq Service$

$S_4 \sqsubseteq =2has.InputParameter$

$S_4 \sqsubseteq =1has.OutputParameter$

$I_{S4} \sqsubseteq InputParameter$

$O_{S4} \sqsubseteq OutputParameter$

ABox

$L : I_{S4}; H : I_{S4}; S : O_{S4};$

$S : Square; L : Verterx; H : Height$

Тобто цей сервіс має *Exact* [1] відповідність запиту за виходами, але взагалі не відповідає запиту за входами.

Подальший пошук дозволив виявити ще два сервіси S_3 та S_2.

Сервіс S_2 має три вхідних параметри, кожний з яких є вершиною трикутника, що задається своїми координатами, а виходом є висота трикутника, яка опущена на одну з сторін.

ТВох для *S2Ontology* може мати вигляд:

TBox

$S_2 \sqsubseteq Service$

$S_2 \sqsubseteq =3has.InputParameter$

$S_2 \sqsubseteq =1has.OutputParameter$

$I_{S2} \sqsubseteq InputParameter$

$O_{S2} \sqsubseteq OutputParameter$

ABox

$A : I_{S2}; B : I_{S2}$

$C \sqsubseteq I_{S2}; hC \sqsubseteq O_{S2}$

*A : Vertex, B: Vertex, C: Vertex /**

Зв‘язок з онтологією *POGeometry*

hC : Height

Сервіс S_3 має два вхідних параметри, кожний з яких – це точка на площі, що задається своїми координатами, а виходом є довжина відрізку між ними. ТВох для *S3Ontology* може мати вигляд:

TBox

$S_3 \sqsubseteq Service$

$S_3 \sqsubseteq =2has.InputParameter$

$S_3 \sqsubseteq =1has.OutputParameter$

$I_{S3} \sqsubseteq InputParameter$

$O_{S3} \sqsubseteq OutputParameter$

ABox

$A : I_{S3}; B : I_{S3}; l_{AB} : O_{S3}$

*A: Vertex, B:Vertex /** Зв’язок з онтологією *POGeometry*

l_{AB} *: LenthAB*

Тобто об’єднання виходів цих двох сервісів – це точна відповідність входу сервісу S_4

$$I_{S4} \equiv O_{S3} \sqcup O_{S2}.$$

Але їх входи досі не забезпечуються запитом. Тобто не виконується умова:

$$I_{S2} \sqcup I_{S3} \sqcup I_{S4} \sqsubseteq I_Q.$$

Таким чином, треба знайти сервіс S такий, що

$$I_{S2} \sqcup I_{S3} \sqsubseteq O_S \ ma \ \ I_S \sqsubseteq I_Q.$$

Нарешті, виявляємо сервіс S_1 із вхідними параметрами *a, b, c* – це вектори, та вихідні параметри *A, B, C*, що є вершинами трикутника. Тобто сервіс, який виконує перетворення з представлення трикутника трьома векторами, у представлення вершинами.

ТВох для S_1*Ontology* може мати вигляд:

$S_1 \sqsubseteq Service$

$S_1 \sqsubseteq =3has.InputParameter$

$S_1 \sqsubseteq =3has.OutputParameter$

$I_{S1} \sqsubseteq InputParameter$

$O_{S1} \sqsubseteq OutputParameter$

<u>ABox</u>

$a : I_{S1}$

$b : I_{S1}$

$c : I_{S1}$

$A : O_{S1}, B : O_{S1}, C : O_{S1}$

*A : Vertex, B: Vertex, C: Vertex /**

Зв'язок з онтологією *POGeometry*

a : Vector, b:Vector, c:Vector

При чому:

$I_Q \sqsubseteq I_{S1}$

$I_{S2} \sqcup I_{S3} \sqsubseteq O_{S1}$ *та* $O_{S1} \sqsubseteq I_{S2} \sqcup I_{S3}$

Тобто графічно можна представити ланцюжок, який показано на рис. 3.

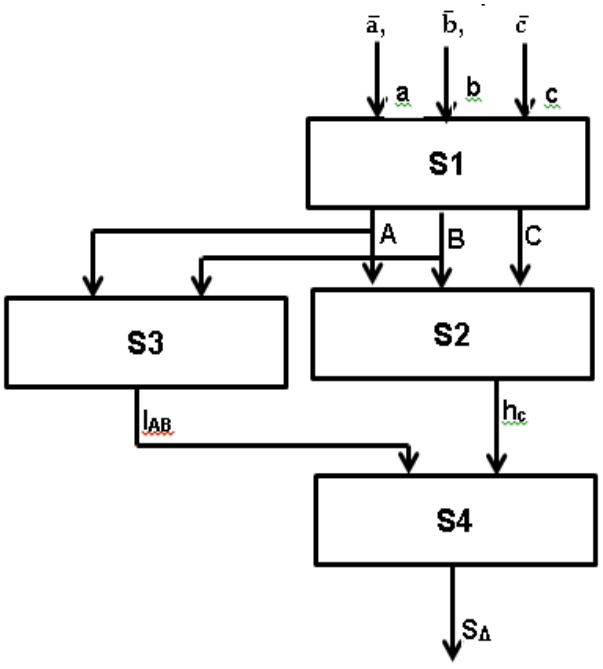

Рис. 3. Ланцюжок побудови композитного сервісу знаходження площі трикутника

Онтологія композиції

Визначимо *загальну онтологію композиції сервісів CompTaskOntology*:

$Q \sqsubseteq Service$

$InputService \sqsubseteq Service$

$OutputService \sqsubseteq Service$

$ServiceComposition \sqsubseteq$

$(hasInput.InputService \sqcup$

$hasInput.Q) \sqcap$

$(hasOutput.OutputService)$

$MatchingXY(S_1,S_2) \sqsubseteq$

$\exists x.Os1, \exists y.Is2 : (C1(x) \sqcap C2(y) \sqcap$

$((C1 \sqsubseteq C2) \sqcap (C2 \sqsubseteq C1)) \sqcup (C1 \sqsubseteq C2))$

Тоді

$ComposedServices(S_1,S_2) \sqsubseteq$

$ServiceComposition \sqcap$

$MatchingXY(S_1,S_2)$

Слід зазначити, що всі наведені вище постановки задач та алгоритми їх вирішення припускають, що сервіси репозиторію та запит мають схоже представлення та посилаються на єдине представлення моделі домена чи онтологію. Але у загальному випадку це досить суворе обмеження, яке суттєво звужує область пошуку. В дійсності існує проблема гетерогенності представлення, як множини сервісів, так і запитів. Запропонована методологія зводить вирішення задачі гетерогенності до задачі інтеграції онтологій, а задачі виявлення та композиції сервісів до класичних алгоритмів ДЛ.

Висновки

Без побудови теоретично обґрунтованої семантичної моделі веб-сервісу та розробки на її основі формалізованих методів та підходів до вирішення основних задач веб-сервісів неможливе подальше ефективне автоматизоване вирішення складних бізнес-задач. В цілому, створений теоретичний апарат є основою для розробки алгоритмів автоматизованого вирішення задач виявлення та композиції семантичних веб-сервісів.

Специфіка запропонованих підходів полягає у тому, що для формального представлення веб-сервісів пропонується використовувати дескриптивну логіку. Доцільність використання дескриптивної логіки обумовлюється тим, що це формалізм, який забезпечує хороше семантичне анотування сервісів. Так як сервіси можуть

створювати змінення у світі, точне представлення їх функціональності повинно мати справу з динамічним аспектом (поведінкою). Це також є одним з ключових аспектів, що робить очевидною доцільність використання ДЛ для описання сервісів, так як їх забезпечує можливість міркувань для сервісів, і таким чином дозволяє описати ДЛ динамічний аспект. Але дослідження сервісів на рівні їх поведінкової, або процесної моделі, є напрямком подальших досліджень. Наявність механізмів міркувань ДЛ забезпечує також можливість автоматизованого виявлення відповідних сервісів та побудови складного сервіса, який реалізує конкретний бізнес-процес.

Формалізація таких загальних понять як веб-сервіс, задача виявлення, композиція за допомогою ДЛ онтолгій є основою вирішення всієї низки задач веб-сервісів на всіх етапах їх життєвого циклу. На основі цих базових визначень побудовані запропоновані методичні підходи, що фактично визначають покроковий ітераційний алгоритм побудови композитного сервісу на функціональному рівні.

Література

1. Захарова О. Визначення та вирішення задачі виявлення Веб-сервісів за допомогою апарату дескриптивних логік. Проблеми програмування. 2017. № 4. С. 66–78.
2. Hai Wang, Zengzhi Li. A Semantic Matchmaking Method of Web Services Based On SHOIN+(D). Institute of Computer System Structure and Networks School of Electronics & Information Engineering, Xi'an Jiaotong University, Xi'an Shaanxi 710049, PR China hwang@mailst.xjtu.edu.cn, lzz@mail.xjtu.edu.cn
3. Baader F., Ku¨sters R., and Molitor R. Computing Least Common Subsumer in Description Logics with Existential Restrictions. In T. Dean, editor, Proc. of the 16th Int. Joint Conf. on AI. P. 96–101. M.K, 1999.
4. Franz Baader, Ralf Kiisters, and Ralf Molitor LuFg Theoretische Informatik, RWTH Aachen. Computing Least Common Subsumers in Description Logics with Existential.
5. Franz Baader, Ralf Ku¨sters, and Ralf Molitor. Rewriting Concepts Using Terminologies. In Proc. of the Int. Conf.KRColorado. USA. P. 297–308, Apr. 2000.
6. Beeri C., Levy A.Y., and Rousset M-C. Rewriting Queries Using Views in Description Logics. In L. Yuan, editor, Proc. of the ACM PODS , New York, USA. P. 99–108, Apr. 1997.
7. Alon Y. Halevy. Answering queries using views: A survey. VLDB Journal, 10(4):270 – 294, 2001.
8. Teege G. Making the difference: A subtraction operation for description log- ics. In J. Doyle, E. Sandewall, and P. Torasso, editors, KR'94, San Francisco, CA, 1994. Morgan Kaufmann.
9. AND/OR Graph and Search Algorithm for Discovering Composite Web Services. Qianhui Althea Lang, Singapore Management University, Stanley Y.W. Su, University of Florida, USA/ http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.1053
10. 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
11. 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
12. http://life-prog.ru/view_zam2.php?id=204&cat=5&page=13
13. Srividya Kona, Ajay Bansal, M. Brian Blake, Gopal Gupta. Generalized Semantics-based Service Composition. / http://ieeexplore.ieee.org/abstract/document/4670179
14. Paolucci M. et al., Semantic Matching of Web Services Capabilities, In First International Semantic Web Conference, Sardinia, Italy. 2002. P. 333–347.

15. https://vo.homelinux.org/wiki/code/FloydWarshall.
16. Cormen T.H. et al. Introduction to Algorithms, MIT Press, 1990.

References

1. *Zakharova O.* Defining and resolving Web-services discovery problems using description logics formalism. Problems in programming. 2017. N 4. P. 66–78.
2. Hai Wang, Zengzhi Li. A Semantic Matchmaking Method of Web Services Based On SHOIN+(D). Institute of Computer System Structure and Networks School of Electronics & Information Engineering, Xi'an Jiaotong University, Xi'an Shaanxi 710049, PR China hwang@mailst.xjtu.edu.cn, lzz@mail.xjtu.edu.cn
3. Baader F., Ku¨sters R., and Molitor R. Computing Least Common Subsumer in Description Logics with Existential Restrictions. In T. Dean, editor, Proc. of the 16th Int. Joint Conf. on AI. P. 96–101. M.K, 1999.
4. Franz Baader, Ralf Kiisters, and Ralf Molitor LuFg Theoretische Informatik, RWTH Aachen. Computing Least Common Subsumers in Description Logics with Existential.
5. Franz Baader, Ralf Ku¨sters, and Ralf Molitor. Rewriting Concepts Using Terminologies. In Proc. of the Int. Conf.KRColorado. USA. P. 297–308, Apr. 2000.
6. Beeri C., Levy A.Y., and Rousset M-C. Rewriting Queries Using Views in Description Logics. In L. Yuan, editor, Proc. of the ACM PODS , New York, USA. P. 99–108, Apr. 1997.
7. Alon Y. Halevy. Answering queries using views: A survey. VLDB Journal, 10(4):270 – 294, 2001.
8. Teege G. Making the difference: A subtraction operation for description log- ics. In J. Doyle, E. Sandewall, and P. Torasso, editors, KR'94, San Francisco, CA, 1994. Morgan Kaufmann.
9. AND/OR Graph and Search Algorithm for Discovering Composite Web Services. Qianhui Althea Lang, Singapore Management University, Stanley Y.W. Su, University of Florida, USA/ http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.1053
10. 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
11. 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
12. http://life-prog.ru/view_zam2.php?id=204&cat=5&page=13
13. Srividya Kona, Ajay Bansal, M. Brian Blake, Gopal Gupta. Generalized Semantics-based Service Composition. / http://ieeexplore.ieee.org/abstract/document/4670179
14. Paolucci M. et al., Semantic Matching of Web Services Capabilities, In First International Semantic Web Conference, Sardinia, Italy. 2002. P. 333–347.
15. https://vo.homelinux.org/wiki/code/FloydWarshall.
16. Cormen T.H. et al. Introduction to Algorithms, MIT Press, 1990.

Одержано 12.09.2017

Про автора:

Захарова Ольга Вікторівна,
кандидат технічних наук,
старший науковий співробітник.
Кількість наукових публікацій в
українських виданнях – 26.
http://orcid.org/0000-0002-9579-2973.

Місце роботи автора:

Інститут програмних систем
НАН України,
проспект Академіка Глушкова, 40.
Тел.: 526 5139.
E-mail: ozakharova68@gmail.com.