Виявлення Веб-сервісів на рівні процесу на основі семантичної анотації
(Переклад Ремарович С.)

Переклад статті «**Web Service Discovery at Process-level Based on Semantic Annotation**»
Francesco Pagliarecci, Marco Pistore, Luca Spalazzi and Paolo Traverso

Абстракт

У даній роботі пропонується новий підхід до виявлення (discovery) розподілених процесів, описаних як семантичні Веб-сервіси. В існуючих підходах виявлення здійснюється за допомогою ключових слів, які пошуковий механізм використовує для класифікації та вибору Веб-сервісів, або на основі "функціонального" опису сервісу з точки зору його входів, виходів, передумов і наслідків. Проте, цей спосіб виявлення Веб-сервісів не завжди є задовільним. Справді, користувач може мати вимоги до поведінки Веб-сервісів. У даній роботі визначено формальну структуру, яка дозволяє виявлення сервісів на рівні процесу, тобто, виявлення сервісів з урахуванням вимог до поведінки сервісу. Підхід заснований на представленні сервісів на рівні процесу, яке базується на специфікації BPEL для поведінки, і який розширює BPEL специфікацію семантичними анотаціями, що дозволяють виконувати обмежене, але ефективне і корисне, семантичне міркування для виявлення Веб-сервісів.

1 Вступ

Багато робіт було виконано з проблеми виявлення сервісів, описаних на *функціональному рівні*, тобто сервісів, описаних як (один запит-відповідь) атомарні компоненти, які, враховуючи деякі входи, повертають деякі виходи. У *задачі виявлення на функціональному рівні*, семантичні Веб-сервіси описуються з входами, виходами, передумовами і наслідками [8, 5]. Проте, визнано, що складні сервіси повинні бути описані також на *рівні процесу* [4, 12, 1], тобто з описом потоку поведінки, необхідного для взаємодії з сервісом. Дійсно, найчастіше, компонентні сервіси не атомарні, і вони взагалі не можуть бути виконані в один крок запит-відповідь. Замість цього вони є процесами станів, які вимагають слідувати протоколу взаємодії, які можуть включати в себе різні послідовні, умовні і ітераційні кроки.

Для цих видів сервісів виявлення на функціональному рівні не може бути повністю задоволеним, так як деякі з вимог, яким повинен задовольняти сервіс, є вимоги до поведінки, яку сервіс повинен проявляти, вони є вимогами про кроки взаємодії бажаного сервісу. Наприклад, розумна вимога процесного рівня для сервісу бронювання готелів є можливість скасувати замовлення після того, як номер був забронювований. Для сервісу електронного уряду в галузі охорони здоров'я вимога може бути в тому, що всі кроки взаємодії, які використовуються, щоб забезпечити особистість і конфіденційність даних, мають бути зроблені після виконання деякої процедури аутентифікації. Вимога до інтернет-магазину може бути в тому, що платіжний реквізит (наприклад, номер кредитної картки) запитується тільки після того, як цикл взаємодії по вибору елементів укладений. Ми називаємо задачу виявлення сервісу, яка включає вимоги до поведінки сервісу, який буде виявлено, *проблемою виявлення рівня процесу* (*process-level discovery problem*).

У цій статті пропонується вирішення проблеми виявлення на рівні процесу, яке грунтується на наступних складових:

– Веб-сервіси описуються на рівні процесу з семантично анотованим абстрактним процесом BPEL. Відповідно до опису в роботі [10], це дозволяє як поведінковий, так і (додатковий) семантичний опис Веб-сервісів.

– Визначається мова, яка може виразити вимоги на поведінку сервісу, який має бути виявлений. Ця мова є темпоральною логікою, яка збагачена твердженнями концептів та ролей.

– Формально визначаємо проблему виявлення сервісів рівня процесу. Поведінка сервісів, які описані як семантично анотовані абстрактні BPEL-процеси, може бути описана як (семантично анотовані) системи перехідних станів (state transition systems). Враховуючи набір сервісів, поведінка яких формалізується анотованими системами перехідних станів, а також враховуючи вимоги виявлення, описані в темпоральній логіці, завдання полягає в тому, щоб знайти такий сервіс, що його поведінка задовольняє вимогу.

– Пропонується новий алгоритм виявлення сервісів, які задовольняють семантично анотовані вимоги поведінки. Алгоритм поєднує в собі міркування про процедурні поведінки, які повинні задовольняти темпоральні умови, а також онтологічні міркування про семантику даних.

Робота побудована таким чином. У розділі 2 описується приклад, що використовується у всій статті. У розділі 3 визначаємо семантично анотовані системи переходів станів, які описують процеси BPEL. У розділі 4 представлено мову для опису семантично анотованих вимог виявлення, і формально визначається задача виявлення на рівні процесу. У розділі 5 описується алгоритм виявлення.

2 Огляд підходу: приклад

Приклад полягає у виявленні існуючих сервісів бронювання квитка в театр, які відповідають заданим вимогам користувача. Припускаємо, що користувач вже обрав набір найбільш прийнятних сервісів за допомогою методів традиційного виявлення функціонального рівня (наприклад, на основі метчінгу за ключовими словами, семантичного метчінгу на основі опису входів-виходів-передумов-ефектів сервісу і т.д.). Тепер користувач повинен вибрати Веб-сервіс відповідно до його опису на рівні процесу.

Згідно підходу [10], кожен Веб-сервіс визначає: онтологію, яка встановлює відповідну термінологію; інтерфейс процесу, який визначає взаємодії, необхідні для виконання сервісу; анотацію хореографії, яка встановлює (частково) відповідності між онтологією та процесом. Надалі, припускаємо, що онтології, які надаються різними сервісами, були відображені в загальну онтологію, яка визначає всі відповідні концепти сценарію виявлення - література з онтологій визначає багато потужних методів для досягнення цього відображення серед локальних онтологій процесів. Будемо вважати також, що процеси інтерфейсу визначаються в абстрактному BPEL і анотовані відповідно до спільної онтології, яка використовує стандартні нотації дескриптивної логіки.

$$
\begin{aligned}
&\mathsf{Time} = \forall\, \mathsf{hh.Number} \sqcap \forall\, \mathsf{mm.Number}\,, && \mathsf{TransStatus}\,,\ \mathsf{PayStatus} \sqsubseteq \top\,,\\
&\mathsf{Date} = \forall \mathsf{year.Number} \sqcap \forall \mathsf{month.Number} \sqcap \forall \mathsf{day.Number}\,, && \mathsf{Gender} \sqsubseteq \top\,,\\
&\mathsf{Client} = \forall \mathsf{clientName.String} \sqcap \forall \mathsf{gender.Gender}\,, && \mathsf{SessionID} \sqsubseteq \mathsf{Number}\,,\\
&\mathsf{Ticket} = \forall \mathsf{sessionID.SessionID} \sqcap \forall \mathsf{ticketID.String} \sqcap \forall \mathsf{show.Show} \sqcap \forall \mathsf{client.Client} \sqcap\\
&\qquad \forall \mathsf{sector.String} \sqcap \forall \mathsf{seatNumber.String} \sqcap \forall \mathsf{status.TransStatus} \sqcap \forall \mathsf{payment.Payment}\,,\\
&\mathsf{Show} = \forall \mathsf{title.String} \sqcap \forall \mathsf{date.Date} \sqcap \forall \mathsf{time.Time} \sqcap \forall \mathsf{theatre.Theatre}\,,\\
&\mathsf{Theatre} = \forall \mathsf{thName.String} \sqcap \forall \mathsf{loc.String}\,,\\
&\mathsf{Payment} = \forall \mathsf{sessionID.SessionID} \sqcap (\leq 1\mathsf{id}) \sqcap (\geq 1\mathsf{id}) \sqcap \forall \mathsf{date.Date} \sqcap \forall \mathsf{amount.Number} \sqcap\\
&\qquad \forall \mathsf{method.PayMethod} \sqcap \forall \mathsf{status.PayStatus}\,,\\
&\mathsf{PayMethod} \sqsubseteq \top, && \mathsf{Cash} \sqsubseteq \mathsf{PayMethod}\,,\\
&\mathsf{CreditCard} = \mathsf{PayMethod} \sqcap \forall \mathsf{client.Client} \sqcap \forall \mathsf{cardType.String} \sqcap \forall \mathsf{cardNumber.String} \sqcap\\
&\qquad \forall \mathsf{from.Date} \sqcap \forall \mathsf{to.Date}\,,\\
&\mathsf{BankTransfer} = \mathsf{PayMethod} \sqcap \forall \mathsf{client.Client} \sqcap \forall \mathsf{bankName.String} \sqcap \forall \mathsf{account.String}\,,\\
&\mathsf{PayPal} = \mathsf{PayMethod} \sqcap \forall \mathsf{client.Client} \sqcap \forall \mathsf{ppAccount.String}
\end{aligned}
$$

Рисунок 1 – Термінологія прикладу

male : Gender, female : Gender, available : TransStatus, notAvailable : TransStatus,
booked : TransStatus, cancelled : TransStatus, waiting : PayStatus,
sentData : PayStatus, authorized : PayStatus, paid : PayStatus

Рисунок 2 – Спільна частина кожного ABox у прикладі

Ця онтологія містить деякі загальні, незалежні від домену концепти (Data, Time, Client), і деякі концепти, які є специфічними для даного домену (Ticket, Show, Payment, ...). Зверніть увагу, що у визначенні Ticket та Payment ми маємо роль status, значення якої обмежені концептами TransStatus і PayStatus відповідно. Ця роль захоплює різні стани запиту клієнта і процедури оплати. Точніше, статус (запитаного) квитка може приймати значення available або NotAvailable залежно від наявності, статус приймає значення booked або cancelled, якщо квиток був замовлений або скасований. Аналогічно, статус оплати є sentData після того, як інформація, пов'язана з платежем (наприклад, номер рахунку або номер кредитної картки), була відправлена, є authorized, якщо клієнт уповноважив платіжний шлюз, і paid, коли платіж буде оброблено, waiting у всіх інших випадках. Можливі значення (екземпляри) для концептів TransStatus і PayStatus перераховані в ABox на рис. 2.

У даному підході процеси інтерфейсу, що визначають поведінку взаємодії компонентних сервісів, визначаються в абстрактному BPEL. BPEL [1] забезпечує операційний опис (динамічної) поведінки Веб-сервісів на верхній частині інтерфейсів сервісів, які визначені в специфікації WSDL. Абстрактний опис BPEL визначає партнерів сервісу, його внутрішні змінні і операції, які спрацьовують при виклику сервісу деякими з партнерів. Операції включають в себе призначення змінних, викликання інших сервісів та отримання відповідей, породження паралельних потоків виконання і недетермінованого вибору однієї дії серед різних напрямів. Стандартні конструкції такі, як if-then-else, case вибори і цикли також підтримуються.

```
<process abstractProcess=”yes” name=”TicketTwo” ... >
...
        <assign name=”offerDataMapping”>
          <copy>
            <!--semann=”/theatreOfferMsg/t:Ticket, tid:String, s:Show, c:Client, sect:String,
        seat:String, p:Payment, today:Date, th:Theatre, cc:CreditCard,
        /theatreOfferMsg/t.sessionID=/theatreRequestMsg/key,
        /theatreOfferMsg/t.ticketID=tid, /theatreOfferMsg/t.show=s,
        /theatreOfferMsg/t.client=c, /theatreOfferMsg/t.sector=sect,
        /theatreOfferMsg/t.seatNumber=seat, /theatreOfferMsg/t.status=available,
        /theatreOfferMsg/t.payment=p, s.title=/theatreRequestMsg/title,
        s.date=/theatreRequestMsg/date, s.time=/theatreRequestMsg/time, s.theatre=th,
        c.name=/theatreRequestMsg/clientName, p.sessionID=/theatreRequestMsg/key,
        p.date=today, p.amount=/theatreRequestMsg/ticketPrice, p.method=cc,
        p.status=waiting, th.thName=/theatreRequestMsg/theatreName” -->
                <from opaque=”yes”/>
                <to part=”ticket” variable=”theatreOfferMsg”/>
         </copy>
        </assign>
...
</process>
```

Рисунок 3 – Фрагмент анотованого BPEL процесу сервісу TicketTwo
[http://www.diiga.univpm.it/~spalazzi/reports/TicketTwo.bpel]

На рис. 3 представлено абстрактну BPEL специфікацію можливого інтерфейсу для сервісу квитків, а саме сервіс TicketTwo. Після отримання запиту на квиток в театр (операція receive на початку файлу), сервіс процесу приймає рішення про наявність квитків (операція switch названа Availability). Зауважимо, що реалізація цієї перевірки, яка залежить від інформаційної системи театру, не має відношення до опису протоколу взаємодії і, отже, не зазначена в абстрактному BPEL. Якщо квиток не доступний (case названа NotAvailableTickets), це доводиться до клієнта і сервіс закінчує виконання. Якщо квиток доступний (case названа AvailableTickets), сервіс відправляє пропозицію і, в разі, якщо клієнт приймає її (OnMessage операція transactionAck), він виконує оплату з кредитної картки.

BPEL специфікація, така як на рис. 3, описує дуже детально шлях взаємодії, який необхідно проводити з Веб-сервісом для того, щоб використовувати його. Тим не менш, цього ще не достатньо, щоб дозволити автоматичне виявлення таких Веб-сервісів. Дійсно, необхідно описати і "семантичні" аспекти таких взаємодій. Ми робимо це за рахунок розширення BPEL специфікації "семантичними анотаціями" (semann атрибути на рис. 3). У прикладі необхідно в першу чергу асоціювати концепти в онтології до (частини) вхідних і вихідних повідомлень, якими обмінюється процес. Це є роль, наприклад, семантичних анотацій "/theatreRequestMsg/ClientName : String, /theatreRequestMsg/theatreName : String, ..." активності receive для операції theatreRequest на початку BPEL процесу. Крім того, необхідно виразити "семантичні" відношення між значеннями вхідних та вихідних даних, переданих під час взаємодії з Веб-сервісом, наприклад, між заголовком шоу, датою і часом, запитаних клієнтом, і квитком, повернутим сервісом бронювання. Це робиться в анотації "... /TheatreOfferMsg/t.show=s, ... s.title=/theatreRequestMsg/title, s.date=/theatreRequestMsg/date, s.time=/theatreRequestMsg /time ... " непрозорого призначення. Подальше використання семантичних анотацій полягає у визначенні результату взаємодії з Веб-сервісом. У нашому прикладі видно, що шоу було замовлено тільки тоді, якщо є квиток, сервіс відправляє пропозицію і користувач підтверджує прийняття пропозиції. Щоб виразити це в специфікації BPEL, ми додаємо анотацію "/theatreOfferMsg/t.status=booked" на активність, яка відповідає отриманню підтвердження.

Зауважимо, що процес, описаний на рис. 3, є лише одним з можливих інтерфейсів для резервування квитків. Різні продавці можуть приймати різні підходи, наприклад, вимагати, щоб інформація про кредитну картку клієнта надавалась перед підтвердженням, а не після, або надання користувачу можливості другого вибору, якщо перша пропозиція була відхилена. Семантичні анотації є необхідними, щоб вирівняти специфікації інтерфейсу вручну, і виразити у відношенні із загальною онтологією. Зауважимо, однак, що семантичні анотації, які повинні бути додані з цією метою, дуже обмежені в порівнянні з процесами, які визначені у таких мовах, як OWL-S або WSMO. Як ми побачимо, вони достатні для задачі автоматичного виявлення, в якій ми зацікавлені.

3 BPEL процеси як анотовані STSs

Ми кодуємо BPEL процеси (розширені з семантичними анотаціями) як *анотовані системи перехідного стану* [10]. Системи перехідного стану (STS) описують динамічні системи, які можуть перебувати в одному зі своїх можливих *станів* (деякі з яких позначені як *початкові стани*) і можуть розвиватися до нових станів в результаті виконання деяких дій.

PROCESS TicketTwo;
STATE *pc* : *{ START, ReceiveRequest, CheckAvailability, AvailableTickets, PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, NotAvailableTickets, invokeNotAvailable, END_NA, END_ACK, END_NACK }*;

```
INIT pc = {START};
CONCEPT String; Number; Date; Time; SessionID; Ticket; Show; Client; Payment; Theatre; CreditCard;
INPUT request(theatreRequestMsg); transactionNAckMsg(); transactionAckMsg();
receiveCCData(CCDataMsg);
OUTPUT ticketsNotAvailable(); theatreResponse(theatreOfferMsg); checkCreditCard(checkCCMsg);
TRANS
pc = START -[TAU]-> pc = ReceiveRequest;
pc = ReceiveRequest -[INPUT request(theatreRequestMsg)]-> pc = CheckAvailablility
pc = CheckAvailability -[TAU]-> pc = NotAvailableTickets;
pc = CheckAvailability -[TAU]-> pc = AvailableTickets;
...
ANNOTATION FUNCTION
...
LAMBDA(ReceiveRequest) = LAMBDA(START);
LAMBDA(CheckAvailability) = { /theatreRequestMsg/clientName:String,
                              /theatreRequestMsg/theatreName:String,
                              /theatreRequestMsg/title:String, /theatreRequestMsg/date:Date,
                              /theatreRequestMsg/time:Time, /theatreRequestMsg/ticketPrice:Number,
                              /theatreRequestMsg/sector:Number, /theatreRequestMsg/key:SessionID}
                              U LAMBDA(ReceiveRequest);
...
```

Рисунок 4 – Фрагмент анотованої STS, яка відповідає процесу TicketTwo
[http://www.diiga.univpm.it/~spalazzi/reports/TicketTwo.smv]

Ми розділяємо дії на *вхідні дії*, *вихідні дії* і τ. *Вхідні дії* являють собою прийом повідомлень, *вихідні дії* представляють повідомлення, відправлені до зовнішніх сервісів, а τ є спеціальною дією, яка називається *внутрішньою дією*, що представляє собою внутрішні еволюції, які є невидимими для зовнішніх сервісів. Іншими словами, τ представляє той факт, що стан системи може розвиватися без створення будь-яких вихідних даних і без споживання будь-яких вхідних даних. *Відношення переходу* описує, як стан може розвиватися на основі входів, виходів або внутрішньої дії τ.

В анотованій STS ми пов'язуємо з кожним станом набір *тверджень концептів* і *тверджень ролей*. Це конфігурує стан як assertional компонент (або ABox) системи представлення знань на основі даної дескриптивної логіки, де онтологія грає роль термінологічного компонента (або TBox). Таким чином, *твердження концепту* являють собою формули виду $a : C$ (або $C(a)$) і заявляють, що даний індивід a належить (інтерпретації) концепту C. *Твердження ролі* являють собою формули виду $a.R = b$ (або $R(a,b)$) і заявляють, що даний індивід b є значення ролі R для a. Як наслідок, кожну дію можна розглядати як перехід зі стану, що міститься в ABox, в інший стан, що міститься в іншому ABox.

Визначення 1 (Анотована система переходів (Annotated state transition system)).
Анотована система переходів – це кортеж $\langle \Sigma, \mathsf{T}, \Lambda \rangle$ де:
$\Sigma = \langle S, S^0, I, O, R \rangle$ - це система переходів;
S – кінцевий набір станів;
$S^0 \subseteq S$ – набір початкових станів;
I – кінцевий набір вхідних дій;
O – кінцевий набір вихідних дій;
$R \subseteq S \times (\mathsf{I} \cup \mathsf{O} \cup \{\tau\}) \times S$ – відношення переходу;
T – термінологія анотації (TBox);
$\Lambda : S \rightarrow 2^{\mathsf{A}_\mathsf{T}}$ - функція анотації, де A_T - набір всіх тверджень концептів і тверджень ролей, визначених над T.

Приклад 1. На рис. 4 показано текстовий опис анотованої STS, який відповідає анотованому коду BPEL на рис. 3. Набір станів S (розділ STATE на рис. 4) моделює етапи процесу і еволюцію тверджень поняття і ролі. pc є змінною, яка пробігає набір станів S і, отже, фіксує поточний крок виконання сервісу (наприклад, pc = CheckAvailability, коли він буде готовий перевірити, чи є квиток на необхідне шоу). Набір початкових станів S^0 представлено розділом INIT на рис. 4. Концепти, що використовуються в анотованій STS, перераховані в розділі CONCEPT на рис. 4. Вони повинні бути визначені в термінах T. Відповідно до формальної моделі, ми розрізняємо три різні види дій. Вхідні дії I моделюють всі вхідні запити до процесу та інформацію, яку вони приносять (наприклад, request використовується для отримання запиту клієнта). Вихідні дії O моделюють всі дії, які відправляють вихідні повідомлення (наприклад, theatreResponse використовується, щоб запропонувати квиток). Дія τ використовується для моделювання внутрішніх еволюцій процесу, таких як призначення і прийняття рішень. Набір вхідних дій I (вихідні дії O) представлено розділом INPUT (OUTPUT). Еволюція процесу моделюється у розділі TRANS на рис. 4.

Функція анотації Λ (див. розділ ANNOTATION FUNCTION на рис. 4) моделює, як твердження варіюються залежно від станів. Наприклад, LAMBDA(PrepareOffer) = {/theatreOfferMsg/t:Ticket, ..., /theatreRequestMsg/title:String, ..., /theatreOfferMsg/t.show =s, , ..., s.title=/theatreRequestMsg/title, ...} представляє той факт, що стан PrepareOffer містить, наприклад, твердження концепту /theatreOfferMsg/t:Ticket, s:Show і /theatreRequestMsg/title:String (тобто /theatreOfferMsg/t, s та /theatreRequestMsg/title є індивідами, які належать до концептів Ticket, Show і String відповідно), а також твердження ролі /theatreOfferMsg/t.show = s, (роль show індивіда /theatreOfferMsg/t заповнюється індивідом s) і s.title=/theatreRequestMsg/title (роль title індивіда s заповнюється індивідами /theatreRequestMsg/title).

Зауважимо, що кожна клауза TRANS і кожна клауза LAMBDA на рис. 4 відповідає різним елементам у відношенні переходу R і в функції анотації Λ, відповідно. Наприклад, перехід і клауза LAMBDA, описані вище, генерують різні елементи R і Λ, залежно від того, яких індивідів /theatreRequestMsg/title ми матимемо в стані призначення. Що стосується cancel, він був визначений на рис. 2 і, таким чином, він позначає той же індивід у всіх станах (тобто всі ABoxes). З цієї причини ми включили в LAMBDA(START) і поширили в усі інші стани твердження, визначені на рис. 2. Визначення системи переходів станів, представлене на рис. 4, тому є параметричним, враховуючи індивідів, які можуть бути пов'язані з концептами, оголошеними в розділі CONCEPT. Для того, щоб отримати конкретну систему перехідного стану (набір конкретних ABoxes) і застосовувати методи, описані в цій статті, має бути призначений кінцевий набір індивідів цим концептам. Можливий підхід, щоб присвоїти ці індивіди, полягає у тому, щоб визначити відповідні твердження концептів у загальній частині ABoxes (наприклад, частина ABoxes зображена на рис. 2). Інший, кращий метод полягає у використанні засобів рівня знань, таких, як в [9], щоб уникнути явного переліку індивідів.

Як уже зазначалося в [10], коли ми повинні перевірити, чи дане твердження p істинне в даному стані, ми повинні застосувати перевірку екземпляра, що позначається як $\langle T, \Lambda(s)\rangle \models p$. У конкретному випадку, коли ми перевіряємо категоризацію, ABoxes не грають активну роль [11], тому категоризація може бути перевірена не враховуючи, який є поточний стан (тобто поточний ABox). Наприклад, коли ми повинні перевірити $\langle T, \Lambda(s)\rangle \models C \sqsubseteq D$, нам потрібно тільки перевірити $\langle T, \emptyset\rangle \models C \sqsubseteq D$. Крім того, припустимо, що ми використовуємо ALN як дескриптивну логіку і узагальнений ациклічний TBox як T [2]. Ця мова досить виразна, щоб описати нетривіальні приклади, як наприклад продавця квитків. Тим не менш, обчислювальна складність категоризації, враховуючи ациклічну термінологію, є NP-повною, в той час як обчислювальна складність перевірки екземпляра

є P [6], що робить міркування над сервісами легким, і, наприклад, менш складним, ніж ті ж в OWL-S.

4 Проблема виявлення (discovery) Веб-сервісу

У відповідності з нашим підходом, входами для завдання виявлення Веб-сервісу є: (1) загальна онтологія $\langle \mathsf{T}, \mathsf{A} \rangle$; (2) набір анотованих BPEL процесів, або, що еквівалентно, анотованих STS-ів $\langle \Sigma_i, \mathsf{T}, \Lambda_i \rangle$, що визначають доступні сервіси; і (3) вимога виявлення ρ, що формалізує бажані властивості Веб-сервісу, який повинен бути обраний. Входи (1) і (2) мають бути вже описані. Тепер звернемо увагу на визначення вимоги виявлення.

Приклад 2 . Ми хотіли б купити квиток на шоу "Кішки" в даному театрі. Тому, в першу чергу, ми повинні знайти Веб-сервіси з бронювання та купівлі такого роду квитків. Крім того, ми не хочемо надавати дані нашої кредитної картки до затвердження квитка. Таким чином, ми хочемо обмежитися сервісами, які виконують ці вимоги на протокол взаємодії.

Після пошуку на функціональному рівні, ми можемо отримати декілька сервісів, які здатні задовольнити нашу першу вимогу, але ми не в змозі відрізнити ті сервіси, які задовольняють нашу другу вимогу. Дійсно, ми повинні перевірити поведінку сервісу, щоб знати, які сервіси задовольняють цю вимогу.

Для того, щоб виразити вимоги виявлення на рівні процесу, такі, як у попередньому прикладі, введемо варіант CTL (Computational Tree Logic) [7], добре відома темпоральна логіка, яка широко використовується для вираження вимог на поведінку динамічних систем. Точніше, оскільки вимога процесного рівня моделюється як CTL формули, формула може бути обчислена на системі перехідних станів, перевіряючи таким чином, чи STS задовольняє вимогу. Ключова особливість розширення CTL полягає в тому, що воно дозволяє вираження умов на анотованій STS, тобто, можна виразити умови на твердження концепту і ролі.

Визначення 2 (Умова мети (Goal Condition)). Нехай $x : C$ буде твердженням концепту і $x.R = y$ буде твердженням ролі приймаючи до уваги T, тоді умова мети визначається наступним чином:

$$p = x : C \mid x.R = y \mid p\mathsf{OR}p \mid p\&p \mid \mathsf{NOT}\, p \mid \mathsf{AF}\, p \mid \mathsf{AG}\, p \mid \mathsf{EF}\, p \mid \mathsf{EG}\, p \mid \mathsf{AX}\, p \mid \mathsf{EX}\, p \mid \mathsf{A}(p\ \mathsf{U}\ p) \mid \mathsf{E}(p\ \mathsf{U}\ p) \mid \mathsf{A}(p\ \mathsf{B}\ p) \mid \mathsf{E}(p\ \mathsf{B}\ p)$$

CTL є пропозиціональною темпоральною логікою розгалуженого часу (branching-time). Інтуїтивно, відповідно до розширення, умова мети повинна бути перевірена на всіх можливих шляхах обчислення (послідовності станів), виходячи з поточного стану. Що стосується пропозиційних зв'язок NOT і $\&$, вони мають своє звичайне значення заперечення і кон'юнкції. Інші пропозиціональні оператори можуть бути визначені в їх термінах. Що стосується темпоральних операторів (наприклад, AF, EF, AX і т. д.), то вони зберегли той же інтуїтивний сенс, що вони мають в стандартній CTL. Іншими словами, $\mathsf{EX}p$ вірно в стані s тоді і тільки тоді, коли s має принаймні наступника t такого, що p істинно при t. $\mathsf{E}[pUq]$ вірно в стані s тоді і тільки тоді, коли існує шлях, починаючи з s і початковий префікс шляху такий, що q має місце на останньому стані префіксу і p виконується у всіх інших станах уздовж префікса. $\mathsf{EG}\, p$ істинно в стані s, якщо є шлях, що виходить з s, такий, що p має місце в кожному стані на цьому шляху. Щодо тверджень концепту та ролі, які є в умові мети, ми називаємо їх твердженнями концепту мети та твердженнями ролі мети, відповідно. Інтуїтивно, твердження концепту мети позначає типову проблему дескриптивної логіки: *проблема пошуку виводу (retrieval inference problem)*. Нехай $x : C$ позначає твердження концепту мети, проблема пошуку виводу є завданням знаходження для кожного стану s всіх індивідів, згаданих у ABox, $\Lambda(s)$, які є екземплярами концепту C щодо даного TBox T. Неоптимізований алгоритм для пошуку

може бути реалізований шляхом тестування для кожного індивіда, що зустрічається в ABox, чи є він екземпляром концепту C. Після того, як ми отримаємо набір екземплярів $\{a\}$ для твердження концепту мети $x : C$, ми можемо замінити x в умові мети отриманими екземплярами і перевірити умову. Таким чином, умова мети позначає набір специфікацій, які повинні бути перевірені замість однієї.

Формальне визначення можна знайти в [7]. Ми також використовуємо наступні синтаксичні скорочення для CTL формул:

- $AXp \equiv NOT\ EX\ NOTp$, що означає, що p виконано в усіх наступних станах поточного стану (p повинен виконуватись у наступному стані).

- $EFp \equiv E\ [trueUp]$, яка означає, що для деякого шляху існує стан на шляху, в якому має місце p (p можливо в майбутньому).

- $AFp \equiv NOT\ EG\ NOTp$, що означає, що для кожного шляху існує стан на шляху, при якому має місце p (p неминуче в майбутньому).

- $AGp \equiv NOT\ EF\ NOT\ p$, що означає, що для кожного шляху в кожному вузлі на шляху p виконується (p виконується незмінно на всьому шляху).

- $A[pUq] \equiv NOT\ E[(NOTq)\ U\ (NOTp\ \&\ NOTq)]\ \&\ NOT\ EG\ (NOTq)$, яка означає, що для кожного шляху існує початковий префікс шляху такий, що q виконується в останньому стані префіксу і p виконується у всіх інших станах, розташованих уздовж префіксу (p виконується поки не виконається q по всіх шляхах).

- $A[pBq] \equiv NOT\ E[NOTpUq]$, яка означає, що для кожного шляху, якщо q завжди відбувається в майбутньому, цьому строго передує виникнення p.

Крім умови, вираженої в нашому варіанті CTL, вимога виявлення також визначає два набори тверджень концептів. Перший, який ми називаємо *твердження концепту входу* (*input concept assertions*), можна розглядати в якості вхідних параметрів для вимог виявлення, такими як бажане шоу і дати, як можна припустити, що існують в ABox глобальної онтології. Другий набір, який названий *твердження концепту виходу*, описує елементи, які повинні бути повернуті виявленим Веб-сервісом, в нашому випадку квиток і платіж. Надамо формальне визначення вимог виявлення (discovery requirements).

Визначення 3 (Вимога виявлення на процесному рівні (Process-level Discovery Requirement)). Нехай T буде термінологією для проблеми композиції. Вимога виявлення – це кортеж $\rho = \langle i, o, p \rangle$, де:

- i - набір тверджень вхідних концептів для T;

- o - набір тверджень вихідних концептів для T;

- p - умова мети на $\langle T, i\ U\ o \rangle$.

Приклад 3. Вимога виявлення сценарію квитка заснована на наступних твердженнях вхідних концептів: x : Show (шоу, яке клієнт бажає відвідати); y : CreditCard (кредитна карта, яку клієнт бажає використовувати для оплати). Твердження вихідних концептів: z : Ticket являє собою квиток, який повертається продавцем; u : Payment являє собою оплату квитка. Як обговорювалося в прикладі 2, мета вимагає, щоб забронювати квиток для даного шоу:

x:Show & y:CreditCard & z:Ticket & u:Payment

EF(z.status = booked & z.show = x & x.title = ‘‘Cats’’ & z.payment = u & u.method = y) & A(z.status = booked) B (u.status = sentData)

У другому рядку формули виражається той факт, що ми шукаємо сервіс, який згодом може призвести до замовлення шоу "Cats" (оператор EF - зверніть увагу, що шоу може бути заброньовано лише за наявності вільних квитків). Третій рядок висловлює умову, що в будь-якому випадку, ми хочемо, щоб квиток був замовлений, перш ніж ми відішлемо дані кредитної картки (оператор A [- B -]).

Визначення 4 (Задоволення вимог виявлення на процесному рівні (Process-level Discovery Requirement)). Нехай $\Gamma = \langle \Sigma, T, \wedge \rangle$ буде анотована система переходів. Нехай $\rho = \langle i, o, p \rangle$ буде вимога виявлення. Тоді Γ задовольняє ρ, записується $\Gamma \models \rho$, тоді і

тільки тоді, якщо для кожного початкового стану $s \in S^0$ існує специфікація p' умови мети p така, що $\langle \Sigma, T, \Lambda(s) \cup i \cup o \rangle \models p'$.

У вимогах виявлення семантичні анотації, введені в BPEL процеси, відіграють фундаментальну роль для визначення умов на результати виконань Веб-сервісу. Тепер, коли ми визначили всі входи виявлення, ми готові надати формальне визначення проблеми виявлення.

Визначення 5 (Проблема виявлення на процесному рівні (Process-level Discovery Problem)). Нехай $\Gamma_1, ..., \Gamma_n$ - набір анотованих систем переходів на тій же термінології T, і нехай $\rho = \langle i, o, p \rangle$ буде вимога виявлення. Проблема виявлення для $\Gamma_1, ..., \Gamma_n$ і ρ - це завдання знаходження анотованої системи переходів із стану в стан Γ_i такої, що $\Gamma_i \models \rho$.

5 Підхід виявлення Веб-сервісу

Згідно з Визначенням 5, *проблема виявлення сервісів рівня процесу може бути зведена до задачі перевірки моделі*: враховуючи набір анотованих систем переходів і вимоги виявлення, основна ідея складається з перевірки моделі умови мети для кожної анотованої STS. Однак, традиційні програми перевірки моделі не можуть бути використані, так як вони не в змозі впоратися з онтологічним міркуванням, необхідним, щоб прив'язати анотації до станів STS. У цьому розділі розглянемо підхід, який дозволяє повторно використовувати якомога більше існуючих програм перевірки моделі (наприклад, NuSMV [3]), для того, щоб експлуатувати дуже ефективні і оптимізовані їх методи перевірки. Цей підхід заснований на ідеї, що треба вирішити *до завдання перевірки моделей* проблему знання, в яких станах твердження, що містяться в умові мети, виконуються. Тобто, перед застосуванням традиційного алгоритму перевірки моделі, ми повинні відобразити онтологічні твердження (анотацій процесу) у висловлювання (propositions). Крім того, ми повинні перетворити твердження, що містяться в умові мети, в умови на висловлювання стану (state propositions). Це досить просто отримати, застосовуючи сервіс пошуку висновків ALN (retrieval inference service of ALN) для кожного твердження концепту мети умови мети та для кожного стану анотованої STS. Тому, для кожного твердження концепту мети, ми повинні набір тверджень концептів відобразити у пропозиції. Після цього ми генеруємо набір тверджень ролі для кожного твердження ролі мети і для кожного знайденого екземпляра. Тоді ми можемо застосувати сервіс перевірки екземпляра з ALN для кожного сформованого твердження ролі і для кожного стану анотованої STS. Кожне твердження ролі, що має місце в даному стані, відображається в судження (proposition) також. Щодо складності, проблема пошуку виводу для ALN має поліноміальну складність. Крім того, ми повинні застосувати поліноміальне число часу на перевірку екземпляра, що для ALN має поліноміальну складність. Результатом є те, що процес відображення тверджень в умову мети має поліноміальну складність.

В наступному кроці ми повинні створити набір специфікацій, підставивши отримані екземпляри у твердження концепту та ролі в умові мети. Нарешті, може бути застосований алгоритм перевірки моделі.

Приклад 4. Розглянемо умову мети, описану в Прикладі 3. Застосувавши сервіс пошуку виведення, отримаємо, що твердження концепту мети x : Show може бути відображено в анотацію s : Show; y : CreditCard може бути відображено в анотацію cc; z : Ticket може бути відображено в анотацію /theatreOfferMsg/t; і u : Payment може бути відображено в анотацію p. Застосовуючи сервіс перевірки екземпляра, отримуємо, в яких станах виконуються твердження, представлені в умові мети (табл. 1). Після цього, ми повинні сформувати всі специфікації. У цьому прикладі ми повинні створити тільки можливу специфікацію, а саме наступну специфікацію:

s:Show & cc:CreditCard & z:Ticket & p:Payment

EF(/theatreOfferMsg/t:Ticket.status = booked & /theatreOfferMsg/t:Ticket.show = s &
s.title = ‘‘Cats’’ & /theatreOfferMsg/t:Ticket.payment = p & p.method = cc) &
A(/theatreOfferMsg/t:Ticket.status = booked) B (p.status = sentData)

Ця специфікація була передана до програми перевірки моделей і в результаті задоволена.

Таблиця 1. Стан та твердження після сервісу виведення пошуку і перевірки екземпляра

Assertion	States
s:Show	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
cc:CreditCard	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
/theatreOfferMsg/t:Ticket	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK, NotAvailableTickets, invokeNotAvailable, END_NA
p:Payment	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
s.title = “Cats”	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
p.method = cc	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
p.status = sentData	receiveCCData, invokeCCService, END_ACK
/theatreOfferMsg/t.status = booked	transactionAck, receiveCCData, invokeCCService, END_ACK
/theatreOfferMsg/t.show = s	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK
/theatreOfferMsg/t.payment = p	PrepareOffer, invokeOffer, requestAck, transactionAck, receiveCCData, invokeCCService, END_ACK, END_NACK

Зазначимо, що, крім розглянутого вище підходу, можливі й інші підходи, де інструменти перевірки моделей розширені для того, щоб мати можливість обробляти спочатку онтологічні анотації в станах і формулах. Ми маємо намір вивчити цей підхід в майбутній роботі.

Використання перевірки моделей дозволяє нам мати вимоги на рівні процесів, як було проілюстровано в наведеному вище прикладі. Використання семантичних анотацій дозволяє мати гнучкі стратегії виявлення. Дійсно, ми можемо переміщатися в онтології для того, щоб зробити дану вимогу більш загальною. Наприклад, узагальнення, згідно цієї стратегії, полягає в пошуку в онтології самих конкретних понять, які відносяться до категорії концептів тверджень концептів мети в умові мети, і переформулювати умову, використовуючи ці концепти, як показано в наступному прикладі.

Приклад 5. Мета в прикладі 3 містить наступне твердження концепту y: CreditCard. У термінології рис. 1, існує концепт PayMethod, який включає CreditCard. З урахуванням цього, основна мета може бути узагальнена:

x:Show & y:PayMethod & z:Ticket & u:Payment &
EF(z.status = booked & z.show = x & x.title = ‘‘Cats’’ & z.payment = u & u.method = y) &
A(z.status = booked) B (u.status = sentData)

У результаті цього узагальнення, ми можемо виявити Веб-сервіси, які дозволяють різні методи оплати, а не тільки за допомогою кредитної карти.

Вище згадана стратегія є узагальненням умови мети в тому сенсі, що виявлені Веб-сервіси являють собою супермножину Веб-сервісів, виявлених за вихідної умови. Дійсно, має місце наступний результат:

Теорема 1 (Повнота, приймаючи до уваги виявлення) (Completeness w.r.t. discovery).

Нехай ρ буде вимогою виявлення і ρ_g відповідне узагальнення щодо термінології T. Тоді, якщо $\Gamma \models \rho$, то $\Gamma \models \rho_g$.

6 Схожі роботи та висновки

У цій статті запропоновано нове вирішення проблеми виявлення Веб-сервісів на основі вимог на їх поведінку. Показано, як поведінкові вимоги можуть бути виражені в CTL, відомій темпоральній логіці, збагаченої твердженнями концепту та ролі. Визначено алгоритм, що заснований на методах перевірки моделі, який, враховуючи опис поведінки сервісу, збагачений семантичними анотаціями, та мету виявлення, яка виражає вимоги поведінки, може визначити, чи сервіс задовольняє мету виявлення. Семантично анотований опис поведінки, що використовується в роботі, вже застосовувався в [10] для композиції сервісів. Використання цієї моделі для виявлення, і, зокрема, підхід перевірки моделі для збагаченої CTL являє собою новий вклад цієї статті.

Велика частина робіт по виявленню семантичних Веб-сервісів вирішує проблему на функціональному рівні і не враховує вимоги поведінки і темпоральні. Робота, описана в [13], представляє фреймворк, в якому виявлення сервісів може керуватися поведінковими описами, наприклад, надані як специфікації UML. Тим не менш, ця робота є ортогональною до представленої, оскільки вона пропонує загальний фреймворк, який може підтримувати розробників в пошуку належних сервісів, а не забезпечення автоматизованого способу для виявлення, як запропоновано в цій статті. WSMO [12] платформа визнає важливість інтерфейсів, які описують поведінку сервісів, а також пропонує використовувати посередників, щоб зіставити поведінки сервісу проти цілей (виявлення). Однак, платформа WSMO не передбачає будь-які конкретні методи для автоматичного виявлення, як запропоновано в цій статті. У більш загальному плані, жодна з попередніх робіт не розглядає проблему виявлення семантично анотованих описів BPEL, залежно від темпоральних вимог.

Література

1. T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S.Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.
2. F. Baader and W. Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.
3. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), 2000.
4. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services, 2003.
5. I. Constantinescu, B. Faltings, and W. Binder. Typed Based Service Composition. In *Proc. WWW'04*, 2004.
6. F. M. Donini. Complexity of Reasoning. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, pages 96–136. Cambridge University Press, 2003.
7. E. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*. Elsevier, 1990.
8. M. Paolucci, K. Sycara, and T. Kawamura. Delivering Semantic Web Services. In *Proc. WWW'03*, 2002.
9. M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition ofWeb Services by Planning at the Knowledge Level. In *Proc. IJCAI'05*, 2005.
10. M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *Proc. of the 3rd European Semantic Web Conference (ESWC 2006)*, Budva (Montenegro), 11–14 June, 2006. Springer–Verlag, Berlin, Germany.
11. A. Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues*. Dottorato di Ricerca in Informatica, Universit`a degli Studi di Roma "La Sapienza", Italia, 1994.
12. SDK WSMO working group. The Web Service Modeling Framework - http://www.wsmo.org/.
13. Andrea Zisman and George Spanoudakis. Uml-based service discovery framework. In *Proc. ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 402–414, 2006.