Мінімалістичний підхід до семантичних анотацій для композицій Веб-процесів
(Переклад Ремарович С.)

Переклад статті «**A Minimalist Approach to Semantic Annotations forWeb Processes Compositions»** Marco Pistore, Luca Spalazzi and Paolo Traverso

Абстракт

Композиція Веб-сервісів, які описані як процеси з фіксацією станів (stateful processes), є дуже серйозною проблемою в ряді доменів застосувань, таких як електронний уряд та електронний банкінг. Семантичні описи Веб-сервісів відкривають шлях до автоматизації їх композиції. Проте, поточні підходи до автоматизованої композиції, які використовують семантику, незважаючи на їх виразну силу, являються важкими у використанні на практиці. Вони вимагають дійсно всеохоплюючих і, зазвичай, великих онтологічних описів процесів, а також досить складних (і часто неефективних) механізмів міркувань.

У цій роботі пропонується мінімалістський підхід до семантичної анотації Веб-сервісів, описаних як процеси, такий, що ми можемо виконати обмежене, але ефективне і корисне семантичне міркування для композиції Веб-сервісів на рівні процесу. Ключова ідея полягає в тому, щоб розділити процедурний опис процесів і онтологічні описи, і додати семантичні анотації до процесів. Визначено формальний фреймворк і запропоновано метод, який може використовувати прості механізми міркувань на онтологічному рівні, інтегровані з ефективними механізмами міркування, розробленими для процедурних описів Веб-сервісів.

1 Вступ

Важливість опису Веб-сервісів на *рівні процесів* широко визнана, про що свідчить наявність стандартних мов для опису бізнес-процесів, як BPEL [1], і найбільш популярних стандартів для семантичних Веб-сервісів, як OWL-S [4] і WSMO [22]. В опису на рівні процесу Веб-сервіс не представлений у вигляді «атомарної» компоненти, яка може бути виконана в однин крок. Замість цього, інтерфейс сервісу описує його поведінку, тобто процес, який взаємодіє з іншими сервісами на різних стадіях, і які можуть бути складені з різними конструкціями управління, наприклад, послідовно, умовно і ітеративно. Описи поведінки Веб-сервісів можуть бути дійсно опубліковані на стандартних мовах, наприклад, як *абстрактні специфікації* BPEL, OWL-S *моделі процесу* і WSMO *інтерфейси*. Вони являють собою ключовий елемент для декількох доменів застосувань, де Веб-сервіси пропонуються в якості основи для взаємодії та інтеграції (бізнес) процесів, які поширюються в мережі. Це є випадок, наприклад, декількох застосувань електронного уряду, інтернет-банкінгу та електронної комерції.

Недавнє дослідження фокусується на ключовій проблемі автоматизованої композиції Веб-сервісів, описаних на рівні процесу [10, 7, 3, 20, 15, 13, 14]. Однак, дослідження все ще перебуває на ранній стадії. З одного боку, деякі підходи не пов’язані з семантичними Веб-сервісами, і тим самим не можуть використовувати здатність виконувати міркування про те, що роблять сервіси. Це той випадок методів для композиції процесів BPEL [15, 13], а також теоретичних основ для композиції сервісів, представлених у вигляді кінцевих автоматів [7, 3]. З іншого боку, підходи, які були запропоновані до цих пір, щоб використовувати семантичні описи (див., наприклад, [10, 20, 14, 22]), засновані на тому, що процеси повинні бути описані за допомогою всеосяжних онтологій. Вони мають практичний недолік, який вимагає довгих описів, які займають багато часу і зусиль, і які дуже важко запропонувати на практиці для промислового застосування. Такі семантичні описи Веб-сервісів засновані на виразних мовах, таких як OWL [9] або WSMO [22], які вимагають складного механізму міркувань. Дійсно, наприклад, сім'я мов OWL заснована на дескриптивній логіці SHIQ і SHIOQ , які мають сервіси міркування, які EXPTIME і NEXPTime, відповідно [19].

У цій статті пропонується практичний підхід до композиції семантичних Веб-сервісів, метою якого є забезпечення методів автоматизованої композиції, які використовують обмежену, як

і раніше корисну і задовільну, кількість семантичних міркувань. Ключова ідея полягає у окремому збереженні процедурного опису процесів і онтологічних описів, і додавання семантичних анотацій, які їх пов'язують. По-перше, процедурна поведінка Веб-сервісу описується мовами, які були розроблені для опису процесів. Потім, семантика даних, якими обмінюються процеси, описана в окремій онтологічній мові. Нарешті, два описи пов'язуються семантичними анотаціями поведінкових описів, які зіставляються з онтологічними концептами. Анотації необхідні, щоб надати семантику даним обміну (наприклад, які відношення існують між вхідними даними сервісу і даними, отриманими у відповідь від сервісу), а також для визначення ефектів і результатів виконань сервісу (наприклад, для виявлення успішних виконань сервісу і, щоб відрізнити їх від невдач, і описати наслідки, пов'язані з успішними виконаннями).

Ми застосовуємо цю ідею на прикладі процесів, описаних у BPEL. Ми надаємо семантику абстрактних BPEL процесів в термінах систем перехідних станів (state transition systems). Змінні, які використовуються в повідомленнях, що ними обмінюються процеси BPEL, складають змінні стану пов'язаних систем перехідних станів. Сенс цих змінних визначається функцією анотації, яка відображає їх до онтологічної мови, яка в даній роботі заснована на дескриптивній логіці ALN і узагальненому ациклічному TBox [2]. Враховуючи цей формальний підхід, ми можемо виразити вимоги композиції як *семантичні цілі* (*semantic goals*), тобто вирази у мові, терміни якої посилаються до онтологічних описів. Ми формально визначили проблему автоматизованої композиції з семантичними цілями і пропонуємо метод автоматичної композиції, яка переводить семантичні цілі в заземлені цілі (*ground goals*), тобто цілі, які посилаються до змінних стану процесу. Таким чином, можна використовувати ефективні автоматизовані алгоритми композиції, які були розроблені для не анотованих BPEL композицій [15, 13]. Доведено, що запропонований метод є надійним і повним, тобто гарантовано генерувати тільки рішення і знайти рішення, якщо воно існує, для вихідного завдання з семантичними цілями.

Робота побудована таким чином. У розділі 2 описуємо стандартний приклад, який використовується у статті. У розділі 3 формально визначаємо семантичні анотації для систем перехідного стану, які описують процеси BPEL, і мову для опису семантичних цілей. У розділі 4 формально визначається проблема композиції, в той час як у розділі 5 описуємо метод автоматизованої композиції.

2 Огляд підходу: приклад

Ми прагнемо автоматизованого синтезу нового композитного сервісу, який взаємодіє з набором існуючих компонентних Веб-сервісів для задоволення даної вимоги композиції. Точніше, ми припустимо, що ми вже визначили постачальників сервісів, які будуть об'єднані в композитний сервіс, і в даний час стикаємося з проблемою визначення виконуваного процесу, який може взаємодіяти з цими постачальниками сервісів з метою досягнення вимоги композиції.

Приклад 1. Наш поточний приклад полягає в композиції існуючих транспортних і готельних сервісів, щоб забезпечити сервіс віртуального туристичного агентства (VTA - *Virtual Travel Agency*). VTA відповідає визначенню прийнятного пакету відпустки, відповідно до запитів користувача. Вибір постачальників сервісів може залежати від обмежень, які задані кінцевим користувачем і знаннями в предметній області: наприклад, якщо пункт призначення поїздки є Париж і тривалість - один тиждень, то ми знаємо, що поїздка може відбутися або літаком, або на поїзді (але не кораблем), і що відповідні розміщення в готелі і в гостьових будинках (але не в квартирах). Отже, ми можемо вважати, що ми вибрали чотири підходящих сервіси: *FlightReservation*, *TrainReservation*, *HotelReservation* і *GuesthouseReservation*.

Згідно нашому підходу кожен Веб-сервіс, що використовується в композиції, визначає:

- онтологію визначення відповідної термінології;
- процес інтерфейсу, що визначає взаємодії, необхідні для виконання сервісу;
- анотація хореографії, яка визначає (частково) відповідності між онтологією та процесом.

Надалі, ми припускаємо, що онтології, надані різними сервісами, були відображені в глобальну загальну онтологію, яка визначає всі відповідні концепти сценарію композиції. Будемо також вважати, що процеси інтерфейсу анотовані відповідно до цієї глобальної онтології.

$$
\begin{aligned}
\text{Date} &= \forall \text{year.Number} \sqcap \forall \text{month.Number} \sqcap \forall \text{day.Number} \\
\text{Client} &= \forall \text{name.String} \sqcap \forall \text{gender.Gender} \\
\text{Gender} &\sqsubseteq \top \\
\text{Status} &\sqsubseteq \top \\
\text{Location} &= \forall \text{name.String} \\
\text{Trip} &= \forall \text{id.String} \sqcap (\leq 1\text{id}) \sqcap (\geq 1\text{id}) \sqcap \forall \text{date.Date} \sqcap \\
&\quad \forall \text{start.Location} \sqcap \forall \text{destination.Location} \sqcap \forall \text{pax.Client} \sqcap \\
&\quad \forall \text{status.Status} \\
\text{Accomodation} &= \forall \text{id.String} \sqcap (\leq 1\text{id}) \sqcap (\geq 1\text{id}) \sqcap \forall \text{date.Date} \sqcap \\
&\quad \forall \text{location.Location} \sqcap \forall \text{pax.Client} \sqcap \\
&\quad \forall \text{status.Status} \\
\\
\text{Flight} &= \text{Trip} \sqcap (\leq 1\text{id}) \sqcap (\geq 1\text{id}) \sqcap \\
&\quad \forall \text{seatNumber.String} \\
\text{Train} &= \text{Trip} \sqcap (\leq 1\text{id}) \sqcap (\geq 1\text{id}) \sqcap \\
&\quad \forall \text{seatNumber.String} \\
\text{Hotel} &\sqsubseteq \text{Accomodation} \sqcap \forall \text{roomNumber.String} \\
\text{GuestHouse} &\sqsubseteq \text{Accomodation} \sqcap \forall \text{roomNumber.String}
\end{aligned}
$$

Рисунок 1 – Термінологія виконуваного прикладу

Приклад 2. Загальна онтологія для сценарію композиції VTA, описаного в прикладі 1, зображена на рис. 1, використовуючи стандартне позначення дескриптивної логіки. Ця онтологія містить частину, яка є спільною для домену VTA (*Date, Client, Location, Trip, Accommodation*) і яку можна розглядати як частину знань домену. Вона також містить інші поняття, які є специфічними фактичних Веб-сервісів, які ми збираємося використовувати в композиції (*Flight,Train, Hotel, GuestHouse*) і які можуть бути отримані шляхом відображення локальної онтології кожного Веб-сервісу в загальну онтологію.

Зверніть увагу, що у визначенні *Trip* та *Accommodation* у нас є статус ролі, значеннями яких допускається тільки концепт *Status*. Ця роль фіксує, який поточний стан запиту клієнта. Дійсно, коли поїздка (розміщення) доступні, статус приймає значення *available*; коли поїздка (проживання) не доступні, статус приймає значення *NotAvailable*, і, нарешті, коли поїздка (проживання) замовлені (анульовані), статус приймає значення *booked* (*cancelled*). Можливі значення (екземпляри), концепту *Status* перераховані в ABox на рис. 2.

male : Gender, female : Gender,
available : Status, notAvailable : Status, booked : Status, cancelled : Status

Рисунок 2 – Спільна частина кожного A-Box у виконуваному прикладі

У 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 від виклику сервісу деякими з партнерів. Операції включають в себе призначення змінних, виклик інших сервісів та отримання відповідей, породження паралельних потоків виконання і недетермінірованого вибору одного серед різних напрямів дій. Стандартні необхідні конструкції, як if-then-else, case choices і петлі, також підтримуються.

Приклад 3. На рис. 3 показано абстрактну BPEL специфікацію сервісу *FlightReservation* в сценарії, обговорюваному у прикладі 1. BPEL специфікація була злегка відредагована, щоб поліпшити читаність.

Процес починається з декларації змінних, які використовуються у вхідних/вихідних повідомленнях: req є вхідною змінною, яке вказує початок і призначення локацій і дати польоту; pax визначає докладну інформацію про клієнта, який бронює політ; offer є політ, який запропонований клієнту, в тому числі ідентифікатор польоту і номер місця. MessageType декларація визначає структуру змінної, яка використовується для відправки/прийому повідомлень. Така структура поряд з визначенням MessageType структур, залучених у різних викликах Веб-сервісів, докладно описані в специфікації WSDL, пов'язаної з кодом BPEL, які ми опускаємо за браком місця.

Інша частина абстрактної специфікації BPEL описує потік взаємодії. Сервіс *FlightReservation* активується за запитом клієнта (інструкція receive відповідної операції request). Інформація про бажаний політ, підтверджена клієнтом, зберігається у змінній req.

Залежно від наявності (switch інструкція іменована checkAvailability), провайдер польоту може або відправити відповідь про відмову в проханні (invoke інструкція відповідає операції not_avail), або підготувати і відправити інформацію про конкретний рейс. В останньому випадку, рейс і номер місця визначається і присвоюється змінній offer в твердженні assign, яке назване prepareOffer. Спосіб, яким інформація отримана, не розкрита і не опублікована абстрактним BPEL: джерела даних, які присвоєні змінним за допомогою конструкцій copy, є «opaque (непрозорими)». Механізм непрозорості дозволяє подання зовнішнього світу з абстрактною точки зору бізнес-логіки, яка приховує частину, що дизайнер не має наміру розкривати і що є стійкою до змін по відношенню до фактичного способу, яким чином внутрішня бізнес логіка визначена (наприклад, виклики на певні бази даних компанії резервування квитків).

Як тільки пропозиція була відправлена клієнту (invoke інструкція відповідної операції offer), сервіс *FlightReservation* призупиняється (інструкція pick), очікуючи, коли клієнт або підтвердить прийняття пропозиції польоту (OnMessage специфікація відповідної операції ack; зауважимо, що це повідомлення несе інформацію про клієнта бронювання польоту), або відмовиться від пропозиції (OnMessage специфікація відповідної операції nack). Тільки в першому випадку взаємодія з сервісом є успішною і політ був замовлений.

Зауважимо, що процес, описаний на рис. 3, є лише одним з можливих інтерфейсів. Різні провайдери польотів можуть приймати різні підходи, наприклад, вимагати, щоб інформація про клієнта надавалась на початку процесу, а не під час підтвердження, або надавати користувачу можливість другого вибору, якщо перша пропозиція польоту була відхилена.

```
<process name="FlightReservation">
    <partnerLinks>
        <partnerLink name="client" partnerLinkType="FtRes_PLT" myRole="FtRes_Server"
    partnerRole="FtRes_Client"/>
    </partnerLinks>
    <variables>
        <variable name="req" messageType="flightRequest"/>
            <!-- "req" contains parts "/req/start", "/req/des", and "/req/date" --->
        <variable name="pax" messageType="paxInformation"/>
            <!-- "pax" contains part "/offer/client" -->
        <variable name="offer" messageType="flightOffer"/>
            <!-- "offer" part "/offer/fl" -->
    </variables>
    <sequence name="main">
        <receive operation="request" variable="req" partnerLink="client"
          semann="/req/start : Location, /req/dest : Location, /req/date : Date"/>
        <switch name="checkAvailability">
          <case name="isNotAvailable">
            <invoke operation="not_avail" partnerLink="client" semann="/offer/fl : Flight,
        /offer/fl.status = notAvailable"/>
          </case>
```

```
                <otherwise name="isAvailable">
                    <assign name="prepareOffer">
                    <copy><from opaque="yes" semann="/offer/fl : Flight, /offer/fl.start = /req/start,
                            /offer/fl.destination = /req/dest, /offer/fl.date = /req/date"/>
                            <to variable="offer" part="fl"/></copy>
                    </assign>
                    <invoke operation="offer" inputVariable="offer" partnerLink="client"/>
                    <pick name="waitAcknowledge">
                            <onMessage operation="ack" cariable="pax" partnerLink="client"
                            semann="/pax/client : Client, /offer/fl.pax = /client/pax, /offer/fl.status =
                    booked"/>
                            <onMessage operation="nack" partnerLink="client" semann="/offer/flight.status
                    = cancelled"/>
                    </pick>
                </otherwise>
        </switch>
        </sequence>
</process>
```

Рисунок 3 - Анотований BPEL процес сервісу *FlightReservation*

Специфікація BPEL описує дуже детально взаємодії, які повинні бути проведені з Веб-сервісом, щоб використовувати його. Проте, цього ще не достатньо, щоб дозволити автоматичну композицію такого Веб-сервісу з іншими сервісами. Дійсно, необхідно описати також «семантичні» аспекти таких взаємодій. Це досягається шляхом розширення специфікації BPEL «семантичними анотаціями» (атрибути semann на рис. 3).

У нашому прикладі необхідно в першу чергу пов'язати поняття в онтології з (частини) вхідними і вихідними повідомленнями, якими обмінюється процес. Це роль, наприклад, семантичних анотацій «/req/start : Location, /req/dest : Location, /req/date : Date» діяльності receive для операції request на початку процесу BPEL. Більше того, необхідно виразити «семантичні» відношення між значеннями вхідних і вихідних даних, якими обмінюються при взаємодії з Веб-сервісом, наприклад, між початком локації і місцем призначення та датами, запитаними клієнтом, і рейсом польоту, повернутого сервісом резервування. Це робиться в анотації «/offer/fl.start = /req/start, /offer/fl.destination = /req/dest, /offer/fl.date = /req/date» непрозорого призначення. Подальше використання семантичних анотацій полягає у визначенні результату взаємодії з Веб-сервісом. У нашому прикладі видно, що рейс буде забронюваний тільки тоді, коли він доступний, сервіс бронювання посилає пропозицію і користувач визнає згоду з пропозицією. Щоб виразити це в специфікації BPEL, ми додаємо анотацію «/offer/status = booked» в діяльності, яка відповідає за отримання підтвердження.

Семантичні анотації необхідні, щоб корегувати особливості інтерфейсу вручну, і покласти це у відношенні із загальною онтологією. Зауважимо, однак, що семантичні анотації, які повинні бути додані з цією метою дуже обмежені в порівнянні з процесами, визначеними у таких мовах, як OWL-S або WSMO. Як ми побачимо, вони достатні для завдання автоматизованої композиції, в якій ми зацікавлені.

3 BPEL процеси як анотовані STS-и

Ми кодуємо BPEL процеси (розширені семантичними анотаціями) як *анотовані системи перехідного стану* (*annotated state transition systems - STS*). STS описують динамічні системи, які можуть бути в одному з їх можливих станів (деякі з них позначені як *вихідні стани*), і можуть розвиватися до нових станів в результаті виконання деяких *дій*. Ми розділяємо дії на *вхідні дії*, *вихідні дії* і τ. *Вхідні дії* являють собою прийом повідомлень, *вихідні дії* представляють повідомлення, передані до зовнішніх сервісів, а τ є спеціальною дією, називається *внутрішня дія*, яка представляє внутрішні еволюції, які не видно зовнішнім сервісам. Іншими словами, τ представляє той факт, що стан системи може розвиватися, не виробляючи будь-який вихід і без

споживання будь-якого входу. *Відношення переходу* (*transition relation*) описує, як стан може розвиватися на основі входів, виходів або внутрішньої дії τ. У анотованій STS ми пов'язуємо набір *тверджень концептів* та *тверджень ролі* з кожним станом. Це конфігурує стан як assertional компонент (або ABox) системи подання 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* є 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 знаходиться в ABox, в інший стан, що міститься в іншому ABox.

Визначення 1 Система переходів (state transition system [16]).
Система переходів станів Σ – це кортеж $\langle S, S^0, I, O, R \rangle$, де:
S – кінцевий набір станів;
$S^0 \subseteq S$ – набір початкових станів;
I – кінцевий набір вхідних дій;
O – кінцевий набір вихідних дій;
$R \subseteq S \times (I \cup O \cup \{\tau\}) \times S$ – відношення переходу.

Визначення 2 Анотована система переходів (Annotated state transition system).
Анотована система переходів – це пара $\langle \Sigma, T, \Lambda \rangle$, де:
Σ - це система переходів;
⟨T, Λ⟩ - анотація;
T – термінологія анотації (TBox);
$\Lambda : S \rightarrow 2^{A_T}$ - функція анотації, де A_T - набір всіх тверджень концептів і тверджень ролей, визначених на T.

```
PROCESS FlightReservation;
STATE
        pc : { START, receive_request, checkAvailability, isNotAvailable, isAvailable,
        invoke_not_available, empty_1, prepareOffer, invoke_offer,
        waitAcknowledge, END_NA, END_ACK, END_NACK };
INIT
        pc = {START};
CONCEPT
        Flight; Location; Date; Client; Status;
INPUT
        request(Location, Location, Date); ack(Client); nack();
OUTPUT
        flightOffer(Flight); not_avail();
TRANS
        pc = START -[TAU]-> pc = receive_request;
        pc = receive_request -[INPUT request(req_start,req_dest,req_dat)]-> pc = checkAvailablility
        pc = checkAvailability -[TAU]-> pc = isNotAvailable;
        pc = checkAvailability -[TAU]-> pc = isAvailable;
        pc = isNotAvailable -[TAU]-> pc = invoke_not_available;
        pc = invoke_not_available -[OUTPUT not_avail()]-> pc = END_NA;
        pc = isAvailable -[TAU]-> pc = prepareOffer;
        pc = prepareOffer -[TAU]-> pc = invoke_offer,
        pc = invoke_offer -[OUTPUT offer(offer_fl)]-> pc = waitAcknowledge;
        pc = waitAcknowledge -[INPUT ack(pax_client)]-> pc = END_ACK;
        pc = waitAcknowledge -[INPUT nack()]-> pc = END_NACK;
ANNOTATION FUNCTION
        LAMBDA(checkAvailability) = { req_start : Location,
                                      req_dest : Location,
                                      req_date : Date };
        LAMBDA(END_NA) = { offer_fl : Flight,
                           offer_fl.status = notAvailable } U LAMBDA(checkAvailability);
        LAMBDA(invoke_offer) = { offer_fl : Flight,
                                 offer_fl.date = req_date,
                                 offer_fl.start = req_start,
```

offer_fl.destination = req_dest } U LAMBDA(*checkAvailability*);
LAMBDA(*END_ACK*) = { pax_client : Client
offer_fl.pax = pax_client,
offer_fl.status = booked} U LAMBDA(*invoke_offer*);
LAMBDA(*END_NACK*) = {offer_fl.status = cancelled} U LAMBDA(*invoke_offer*);

Рисунок 4 – Анотована STS, яка відповідає процесу FlightReservation

Приклад 4. 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 концептів і ролі. pc є змінною, яка пробігає набір станів S і, отже, зберігає поточний крок виконання сервісу (наприклад , pc = checkAvailability коли він буде готовий для перевірки, чи політ є доступним). Набір початкових станів S^0 представлено розділом INIT на рис. 4.

Концепти, що використовуються в анотованій STS перераховані в розділі CONCEPT рис. 4. Вони повинні бути визначені в термінології T.

Відповідно до формальної моделі, ми розрізняємо три різні види дій. Вхідні дії I моделюють всі вхідні заявки, що надійшли до процесу, та інформацію, яку вони приносять (наприклад, request використовується для отримання запиту на резервування квитків). Вихідні дії O представляють вихідні повідомлення (наприклад, flightOffer використовується в пропозиції польоту). Дія τ використовується для моделювання внутрішніх еволюцій процесу, таких як призначення і прийняття рішень.

Еволюція процесу моделюється за допомогою набору можливих переходів (розділ TRANS на рис. 4). Кожен перехід визначає його умови застосовності до висхідного стану, його запускаюча дія і стан призначення. Наприклад , *pc* = checkAvailability -[TAU]-> *pc* = isNotAvailable заявляє, що дія τ може бути виконана в стані checkAvailability і призводить до стану isNotAvailable; цей перехід моделює рішення сервісу бронювання, що політ не є доступним.

Функції анотації Λ (див. розділ ANNOTATION FUNCTION на рис. 4) моделює, як твердження змінюються в залежності від станів. Наприклад, *LAMBDA(*END_NACK*)* = {*req_start : Location, req_dest : Location, req_date : Date, offer_fl : Flight, offer_fl.start = req_start, offer_fl.destination = req_dest, offer_fl.date = req_date, offer_fl.status = cancelled*} представляє той факт, що стан END_NACK містить, наприклад, твердження концепту *fl : Flight* (тобто , *fl* є індивід, який належить до концепту *Flight*) і твердження ролей *offer_fl.start = req_start* і *offer_fl.status = cancelled* (ролі *start* і *status* індивіда *fl* заповнені індивідами *req_start* і *cancelled*).

Зауважимо , що кожна клауза TRANS і кожна клауза *LAMBDA* рис.4, відповідає різним елементам у відношенні переходу R і в функції анотації Λ відповідно. Наприклад, клауза переходу і клауза *LAMBDA,* описані вище, генерують різні елементи R і Λ залежно від того, які індивіди *req_date*, *req_start*, *req_dest* ми маємо в стані призначення. Що стосується *cancel*, це було визначено на рис. 2, і, таким чином, він позначає один і той же індивід у всіх станах (тобто, всі ABoxes).

Визначення системи переходу стану, надане на рис. 4, є параметричним щодо індивідів, які можуть бути пов'язані з концептами Location, Date, Client, Flight. Для того, щоб отримати конкретну систему перехідного стану (набір конкретних ABox-ів) і застосувати автоматизовані методи синтезу, описані в цій статті, кінцевий набір індивідів повинен бути призначений концептам Location, Date, Client, Flight. Можливий підхід, щоб призначити ці індивіди, полягає у визначенні відповідних тверджень концептів у загальній частині ABox-ів (наприклад, частина ABox-ів зображена на рис. 2). Інший, кращий метод полягає у використанні методів рівня знань, таких як в [13], щоб уникнути явного перерахування індивідів Location, Date, Client, Flight.

Ми формально визначили переклад, який пов'язує анотовану систему перехідного стану з кожним компонентним сервісом, починаючи з його анотованої специфікації BPEL. На рис. 4 ми вже повідомляли переклад конкретного випадку сервісу з бронювання авіаквитків, з незначними змінами (наприклад, в питанні клауз і в деяких автоматично згенерованих іменах), щоб поліпшити

читаність. Ми опускаємо формальне визначення перекладу, яке можна знайти на http://www.astroproject.org/.

Згідно з наведеними вище визначеннями, коли ми повинні перевірити, чи дане твердження p вірно в даному стані $s \in S$, ми повинні застосувати перевірку екземпляра, позначається як $\langle \mathsf{T}, \Lambda(s) \rangle \models p$. З іншого боку, ABox-и не грають активну роль при перевірці категоризації [17], тому категоризацію можна перевірити не враховуючи поточний стан (тобто, поточний ABox). Наприклад, коли ми повинні перевірити $\langle \mathsf{T}, \Lambda(s) \rangle \models C \sqsubseteq D$, нам потрібно тільки перевірити $\langle \mathsf{T}, 0 \rangle \models C \sqsubseteq D$. Крім того, припустимо, що використовуємо ALN як дескриптивну логіку і узагальнений ациклічний TBox як T [2]. Ця мова досить виразна, щоб описати нетривіальні приклади, як наприклад VTA (Virtual Travel Agency). Тим не менш, обчислювальна складність категоризації щодо ациклічної термінології є NP-повною, в той час як обчислювальна складність перевірки екземпляра є P [6], що робить сервіси міркування розв'язними, і, наприклад, менш складними, ніж в OWL-S.

4 Проблема композиції Веб-процесу

У відповідності з нашим підходом, входами для проблеми композиції є: (1) глобальна онтологія $\langle \mathsf{T}, \mathsf{A} \rangle$; (2) набір анотованих BPEL процесів, або, що еквівалентно, анотованих STS-ів $\langle \Sigma_i, \mathsf{T}, \Lambda_i \rangle$, які визначають компонентні сервіси, і (3) вимога композиції ρ, що формалізує ці бажані властивості композитного сервісу, який буде синтезований. Входи (1) і (2) були вже описані. Тепер увага на визначенні вимоги композиції.

Приклад 5. Ми хочемо, щоб VTA визначала і бронювала пакет відпочинку відповідно до запиту замовника. Це означає, що ми хочемо VTA, щоб досягти ситуації, коли поїздка була заброньована з початку місцеположення до пункту призначення, вказаних користувачем, і для дат, вказаних користувачем, більше того, проживання повинно бути замовлене для того ж пункту призначення і дати. Однак, ця мета VTA може бути неможливою. Може бути неможливо замовити поїздку або житло для даного призначення, або, в більш реалістичних описах VTA, пакет, визначений VTA, може бути занадто дорогим. Ми не можемо уникнути таких ситуацій, і тому ми не можемо попросити композитний сервіс, який гарантує, що пакет відпочинку завжди визначається і замовляється. Тим не менш, ми хотіли б VTA, який спробує (зробити все, що можна), щоб задовольнити це. Більше того, у випадку, коли вимога «визначити і забронювати пакет відпочинку» не виконується, ми не хочемо замовити тільки поїздку або тільки розміщення. Тобто, або проїзд та проживання обидва замовлені, або ніщо з них не повинно бути замовлене. Назвемо цю вимогу «ніяке замовлення не перебуває в очікуванні (*no booking is pending*)». Наша вимога композиції буде виглядати наступним чином:

try to define and book a vacation package;
upon failure, guarantee that no booking is pending.

Зауважимо, що при композиції Веб-сервісів, це часто буває, що вимоги композиції мають структуру, описану в попередньому прикладі, тобто вони визначають «основну умову», яка повинна бути досягнута, коли це можливо, і «умову відновлення (recovery condition)», яка повинна бути досягнута в усіх випадках, коли головна умова не виконується. Крім головної умови і умови відновлення, вимога композиції також визначає два набори тверджень концепту. Перший набір, який ми називаємо *вхідні твердження концептів* (*input concept assertions*), можна розглядати в якості вхідних параметрів для вимог композиції, таких як бажаний пункт призначення поїздки і дати, які можна припустити, що існують в ABox глобальної онтології. Другий набір, званий *вихідні твердження концептів* (*output concept assertions*), описує елементи, які повинні бути визначені композицією Веб-сервісу, в нашому випадку поїздка та розміщення.

Дамо формальне визначення умови мети.

Визначення 3 (Умова мети).

Нехай *a : C* буде твердженням концепту і *a.R = b* буде твердженням ролі стосовно T, тоді умова мети визначається наступним чином:

p = a : C | a.R = b | p OR p | p & p | NOT p

Нарешті, ми можемо дати формальне визначення вимоги композиції.

Визначення 4 (Вимога композиції).

Нехай T буде термінологією проблеми композиції. Вимога композиції – це кортеж $\rho = \langle i, o, p, r \rangle$, де:

- *i* - набір тверджень вхідних концептів для T;
- *o* - набір тверджень вихідних концептів для T;
- *p* - умова мети на $\langle T, i \cup o \rangle$, яка визначає основну умову;
- *r* - умова мети на $\langle T, i \cup o \rangle$, яка визначає умову відновлення.

Приклад 6. Вимога композиції сценарію *VTA* заснована на наступних твердженнях вхідних концептів:

start : Location – описує початкову локацію для подорожі;
dest : Location – описує пункт призначення для подорожі;
date : Date – описує дати подорожі;
client : Client– описує клієнта, який замовляє подорож.

Твердження вихідних концептів наступні:

tr : Trip – поїздка, яка повертається *VTA;*
ac : Accommodation – житло, яке повертається *VTA.*

Як обговорювалось в прикладі 5, основна мета вимагає замовити підходящу подорож і підходяще житло:

tr.date = date & tr.start = start
& tr.destination = dest & tr.pax = client
& tr.status = booked
& ac.date = date
& ac.location = dest
& ac.pax = client & ac.status = booked

Умова відновлення (recovery condition) для вимоги композиції визначає, що ні подорож, ні житло не може бути замовлено, і може виглядати наступним чином:

tr.status ≠ booked & ac.status ≠ booked

У вимогах композиції семантичні анотації, які введені в BPEL процеси, відіграють фундаментальну роль для визначення умов на результати виконання Веб-сервісу. Наприклад, семантичні анотації, які визначають відповідності між вхідними і вихідними повідомленнями сервісу резервування квитків, дають зрозуміти, які значення повинні бути передані до цього сервісу, щоб замовити політ. Більше того, значення, яке присвоєне *offer_fl.status* у фінальних діях різних гілок процесу, дозволяє відрізнити успішні виконання (переліт замовлений) від збоїв (у зв'язку з відсутністю доступних рейсів або скасування резервування).

Тепер, коли ми визначили всі входи композиції, ми готові надати формальне визначення проблеми композиції. Надалі, ми повторно використовуємо визначення, які запропоновані в [15, 16] і адаптовані до випадку анотованих STS.

Перший крок у визначенні проблеми композиції полягає в об'єднанні анотованих STS $\Gamma_i = \langle \Sigma_i, T, \Lambda_i \rangle$, які відповідають різним компонентним сервісам, в єдину STS $\Gamma_{||} = \langle \Sigma_{||}, T, \Lambda_{||} \rangle$, яка визначає комбіновану поведінку компонентних сервісів. Точніше, $\sum_{||} = \sum_1 || \sum_2 || \ldots || \sum_n$ являє собою STS, яка визначає паралельний продукт $\sum_i$, де кожен компонент еволюціонує незалежно від інших, тобто кожен перехід $\sum_{||}$ відповідає переходу одного з

компонентів (див. [15, 16] для формального визначення). Крім того, $\Lambda_{||}$ зіставляє з кожним станом $\sum_{||}$ всі анотації відповідних станів $\sum_1$, $\sum_2$, …, $\sum_n$.

Автоматизований синтез композитного сервісу полягає в створенні нової системи перехідного стану $\sum_c$, яка після підключення до $\sum_{||}$ задовольняє вимогу композиції. Тепер формально визначимо систему переходів станів, яка описує поведінки $\sum$ при підключенні до $\sum_c$.

Визначення 5 (Контрольована система [16]) (Controlled system).

Нехай $\Sigma = \langle \mathcal{S}, S^0, I, O, \mathsf{R}\rangle$ і $\Sigma_c = \langle S_c, S^0_c, I_c, O_c, \mathsf{R}_c\rangle$ - дві системи перехідного стану такі, що $I = O_c$ і $O = I_c$. Система перехідного стану $\sum_c \rhd \sum$, яка описує поведінки системи $\sum$, коли контролюється системою $\sum_c$, визначається наступним чином:

$$\sum_c \rhd \sum = \langle S_c \times S, S^0_c \times S^0, I, O, \mathsf{R}_c \rhd \mathsf{R}\rangle,$$

де:

$\langle (s_c, s), \tau, (s'_c, s)\rangle \in (\mathsf{R}_c \rhd \mathsf{R})$, якщо $\langle s_c, \tau, s'_c\rangle \in \mathsf{R}_c$;

$\langle (s_c, s), \tau, (s_c, s')\rangle \in (\mathsf{R}_c \rhd \mathsf{R})$, якщо $\langle s, \tau, s'\rangle \in \mathsf{R}$;

$\langle (s_c, s), a, (s'_c, s')\rangle \in (\mathsf{R}_c \rhd \mathsf{R})$, при $a \neq \tau$, якщо $\langle s_c, a, s'_c\rangle \in \mathsf{R}_c$ і $\langle s, a, s'\rangle \in \mathsf{R}$.

Це визначення може бути легко розширене на випадок анотованої STS. Дійсно, композитний сервіс $\sum_c$ не має анотацій, і, отже, анотації стану $\sum_c \rhd \Gamma$ являються тими, які відповідають стану в Γ.

У задачі композиції Веб-сервісу ми повинні генерувати $\sum_c$, що гарантує задоволення вимог композиції ρ. Формалізація вимоги, що керована система $\sum_c \rhd \Gamma_{||}$ повинна задовольняти ρ, записується $\sum_c \rhd \Gamma_{||} \models \rho$.

Визначення 6 (задоволення мети) (goal satisfaction).

Нехай $\Gamma = \sum_c \rhd \Gamma_{||}$, і нехай $\rho = \langle i, o, p, r\rangle$ буде вимогою композиції.

Ми говоримо, що Γ строго задовольняє ρ, записано $\Gamma \models_s \rho$, якщо всі кінцеві стани s системи Γ є такими, що $\langle \mathsf{T}, \Lambda(s) \cup i \cup o\rangle \models p$.

Ми говоримо, що Γ слабо (weakly) задовольняє ρ, записано $\Gamma \models_w \rho$, якщо всі кінцеві стани s системи Γ є такими, що $\langle \mathsf{T}, \Lambda(s) \cup i \cup o\rangle \models p$ або $\langle \mathsf{T}, \Lambda(s) \cup i \cup o\rangle \models r$.

Ми говоримо, що Γ задовольняє ρ, записано $\Gamma \models \rho$, якщо:

$\Gamma \models_s \rho$, або

$\Gamma \models_w \rho$ і не існує $\Gamma' = \sum'_c \rhd \Gamma$ такої, що $\Gamma' \models_s \rho$.

Згідно з цим визначенням, керована (controlled) система задовольняє мету композиції, якщо:

або всі стани, які досягнуто в кінці обчислення, задовольняють основну умову (у даному випадку ми говоримо, що мета виконується в строгий спосіб),

або всі стани, які досягнуто наприкінці обчислень, задовольняють або головну умову, або

умову відновлення (в даному випадку ми говоримо, що мета буде задоволена в слабкий спосіб), і ніякий контролер сильного задоволення не може бути визначений (тобто слабке задоволення є найкраще, що ми можемо досягти).

Визначення 7. Проблема композиції (composition problem) [16].

Нехай $\Gamma_1, \ldots, \Gamma_n$ буде набір анотованих систем перехідних станів на тій же самій термінології T і нехай ρ буде вимогою композиції. Проблема композиції для $\Gamma_1, \ldots, \Gamma_n$ і ρ полягає у проблемі знаходження системи перехідного стану $\sum_c$ такої, що $\sum_c \rhd (\Gamma_1 || \ldots || \Gamma_n) \models \rho$.

5 Автоматизований синтез композиції процесу

У роботах [15, 16] описується алгоритм, який автоматично створює композитний сервіс $\sum_c$, починаючи з компонентних сервісів $\sum_1, \sum_2, \ldots, \sum_n$ і вимоги композиції ρ. Крім того, алгоритм був реалізований в наборі інструментів ASTRO (див. http://www.astroproject.org/) і застосований до різних областей, що показує, що він здатний виконувати композицію складних сервісів за дуже невеликий час, набагато менший, ніж час, необхідний для ручної реалізації композитного процесу.

Тим не менш, у роботах [15, 16] компонентні STS-и не використовують семантичні анотації і вимога композиції була заснована на логіці висловлювань замість дескриптивної логіки та онтологій. Наша мета полягає в розширенні підходу робіт [15, 16] до випадку композиції процесу з семантичними анотаціями. Існують різні способи досягнення цієї мети. Ми приймаємо дуже простий підхід, що полягає в перетворенні обмежень вимоги композиції у пропозиціональні формули через процес *заземлення* (*grounding* process). Як тільки це зроблено, семантичні анотації компонентних STS систем можуть бути інтерпретовані як «синтаксичні» анотації, які не підлягають категоризації, і алгоритм [15, 16] можна використовувати повторно. Опишемо детально процес заземлення.

Мета процесу композиції полягає у визначенні відповідним чином вихідних індивідів мети композиції, так щоб основні умови або умови відновлення задовольнялись. Процес заземлення полягає у тому, щоб знайти в онтології всі концепти, які включено до складу концептів вихідних тверджень концептів в меті, і переформулювати умови в меті, використовуючи об'єднання всіх цих концептів для вихідних індивідів, як показано у наступному прикладі.

Приклад 7. Мета в прикладі 6 визначає два вихідних твердження концептів, а саме: tr : Trip і ac : Accomodation. В термінології рис. 1 існують два концепти, які належать Trip, а саме Flight і Train, і два концепти, які включені в категорію Accomodation, а саме Hotel і GuestHouse.

З урахуванням цього, основна мета може бути заземлена наступним чином:

tr: (Trip $\bigsqcup$ Flight $\bigsqcup$ Train)
& tr.date = date & tr.start = start
& tr.destination = dest & tr.pax = client
& tr.status = booked
& ac: (Accommodation $\bigsqcup$ Hotel $\bigsqcup$ GuestHouse)
& ac.date = date
& ac.location = dest
& ac.pax = client & ac.status = booked

Аналогічно, умова відновлення може бути заземлена як:

tr: (Trip $\bigsqcup$ Flight $\bigsqcup$ Train)
& tr.status $\neq$ booked

& ac: (Accommodation $\bigsqcup$ Hotel $\bigsqcup$ GuestHouse)

& ac.status $\neq$ booked

Виконуються наступні результати.

Лема 1.
Нехай $\langle \mathsf{T}, 0 \rangle \models C \sqsubseteq D$ і нехай $p(D)$ буде пропозиціональна формула, яка містить концепт D, тоді
$\langle \mathsf{T}, \Lambda(s) \rangle \models p(D)$ тоді і тільки тоді, коли $\langle \mathsf{T}, \Lambda(s) \rangle \models p(D \bigsqcup C)$
$\langle \mathsf{T}, \Lambda(s) \rangle \not\models p(D)$ тоді і тільки тоді, коли $\langle \mathsf{T}, \Lambda(s) \rangle \not\models p(D \bigsqcup C)$
Доведення. Коли $C \sqsubseteq D$, то D являється еквівалентним до $C \bigsqcup D$. Тому лема доводиться по індукції.

Лема 2.
Нехай $\rho = (p, r)$ буде ціль, де p являється головною умовою і r являється умовою відновлення.
Нехай $\langle \mathsf{T}, 0 \rangle \models C \sqsubseteq D$, тоді
$\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models \rho(D)$, якщо і тільки якщо $\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models \rho(D \bigsqcup C)$.
Доведення. Що стосується головної умови мети ρ, за лемою 1 маємо
$\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models p(D)$, якщо і тільки якщо $\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models p(D \bigsqcup C)$.
Що стосується умови відновлення мети ρ, за лемою 1 маємо
$\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \not\models p(D)$ і $\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models r(D)$,
якщо і тільки якщо
$\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \not\models p(D \bigsqcup C)$ і $\langle \sum, \langle \mathsf{T}, \Lambda(s) \rangle \rangle \models r(D \bigsqcup C)$.

Наслідок 1. Нехай ρ буде вимогою композиції і нехай $\langle \mathsf{T}, 0 \rangle \models C \sqsubseteq D$. Тоді
$\sum_c \rhd (\Gamma_1 || \ldots || \Gamma_n) \models \rho(D)$, якщо і тільки якщо $\sum_c \rhd (\Gamma_1 || \ldots || \Gamma_n) \models \rho(D \bigsqcup C)$.
Доведення. За лемою 2 і визначенням задоволення мети.

Теорема 1 (Коректність і повнота стосовно композиції).
Нехай ρ буде вимогою композиції і ρ_g буде відповідним заземленням стосовно термінології T. Тоді
$\sum_c \rhd (\Gamma_1 || \ldots || \Gamma_n) \models \rho$, якщо і тільки якщо $\sum_c \rhd (\sum_1 || \ldots || \sum_n) \models_g \rho_g$,
де $\models_g$ є здійснимість з умовами мети, інтерпретується як пропозиціональні формули.
Доведення. За наслідком 1 ми знаємо, що $\sum_c \rhd (\Gamma_1 || \ldots || \Gamma_n) \models \rho$, якщо і тільки якщо $\sum_c \rhd (\sum_1 || \ldots || \sum_n) \models \rho_g$. Оскільки ρ_g являється результатом процесу заземлення, здійсниміть на основі онтологічного рівня і пропозиціонального рівня співпадає.

6 Пов'язані роботи та висновки

У даній роботі показано, як, зберігаючи (BPEL) опис поведінки Веб-сервісу, відокремити його від онтологічного опису, і, зв'язавши два описи через семантичну анотацію, ми можемо використовувати ефективні методи для міркувань про поведінку і про семантику сервісів, а також автоматично виконувати композицію сервісів у практичній площині. Дійсно, ми можемо

використовувати методи планування для композиції BPEL процесів, які були показані в масштабі до складних прикладів, як з тематичних досліджень, взятих з реальних додатків, так і параметризованих доменів [15, 13].

З одного боку, деякі роботи запропонували підходи до композиції на рівні процесу, які не розглядають явно необхідність (міркування про) семантичних описів Веб-сервісів [7, 3, 15, 13]. З іншого боку, велика частина робіт по автоматизованій композиції семантичних Веб-сервісів була зосереджена досі на проблемі композиції на функціональному рівні, тобто композиція атомарних сервісів, які можуть бути виконані в одному кроці запит-відповідь [11, 5].

Робота по WSDL-S і Метеор-S [18, 12, 21] забезпечує семантичну анотацію для WSDL. Це близько по духу до нашої, але не має справи з семантично анотованими (BPEL) описами Веб-сервісів рівня процесу. Робота [8] також близька за духом до нашої спільної мети, щоб подолати розрив між фреймворком семантичного вебу і мовами моделювання процесу і виконання, які запропоновані промисловими коаліціями. Тим не менш, робота [8] присвячена іншій проблемі, тобто розширення BPEL семантичними Веб-технологіями для полегшення взаємодії Веб-сервісів, у той час як проблема автоматизованої композиції не вирішена.

Останнім часом все більша кількість робіт має справу з проблемою композиції семантичних Веб-сервісів з урахуванням їх поведінкових описів [10, 23, 20, 14, 22]. У цьому контексті дослідження дотримується двох споріднених, але різних основних підходів: OWL-S [4] і WSMO [22]. Підходи на основі OWL-S [10, 23, 20, 14] відрізняються від запропонованого підходу у даній роботі, так як в OWL-S навіть процеси описуються як онтології, і, отже, немає ніякого способу відокремити міркування про процеси і міркування про онтології. Підхід, який здійснений у WSMO, ближче за духом до наших: процеси представлені у вигляді абстрактних машин станів (Abstract State Machines), добре відомий і загальний формалізм для представлення динамічної поведінки. Ідея, що лежить в основі WSMO, полягає в тому, що змінні абстрактних машин станів всі визначені з точки зору онтологічної WSMO мови. Наші процеси працюють на своїх власних змінних стану, деякі з яких можуть бути відображені в окрему онтологічну мову, що дозволяє мінімалістський і практичний підхід до семантичних анотацій і для ефективного міркування, щоб виконати композицію сервісу автоматично. Справді, мета роботи по WSMO полягає в тому, щоб запропонувати спільну мову і механізм представлення для семантичних Веб-сервісів, в той час як ми концентруємо увагу на проблемі забезпечення ефективних методів для автоматичної композиції семантичних Веб-сервісів.

Було б цікаво дослідити, як наш підхід може бути застосований до WSMO абстрактних машин стану, а не BPEL процесів, і як ідея мінімалістських семантичних анотацій може бути розширена для роботи з мовами WSMO оркестровки і механізмами так, що б ефективно використовувати автоматизовані методи композиції в цьому фреймворку. Це завдання знаходиться в наших наукових дослідженнях. У зв'язку з цим ми плануємо також інтегрувати нашу пропозицію для автоматизованої композиції з методами для виявлення Веб-сервісів, проблеми, яка не розглядається в цій статті.

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 forWeb 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. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of E-Services that export their behaviour. In *Proc. ICSOC'03*, 2003.
4. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services, 2003.
5. I. Constantinescu, B. Faltings, andW. 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. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Behind the Curtain. In *Proc. PODS'03*, 2003.
8. D.Mandell and S.McIlraith. Adapting BPEL4WS for the SemanticWeb: The Bottom-Up Approach to Web Service Interoperation. In *Proc. of 2nd International Semantic Web Conference (ISWC03)*, 2003.
9. D. L. McGuinness and Editors F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

10. S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW'02*, 2002.
11. M. Paolucci, K. Sycara, and T. Kawamura. Delivering Semantic Web Services. In *Proc. WWW'03*, 2002.
12. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web Service Annotation Framework. In *WWW04*, 2004.
13. M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. IJCAI'05*, 2005.
14. M. Pistore, P. Roberti, and P. Traverso. Process-level compositions of executable web services: onthe-fly versus once-for-all compositions. In *Proc. ESWC'05*, 2005.
15. M. Pistore, P. Traverso, and P. Bertoli. Automated Composition of Web Services by Planning in Asynchronous Domains. In *Proc. ICAPS'05*, 2005.
16. M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated Synthesis of Composite BPEL4WS Web Services. In *Proc. ICWS'05*, 2005.
17. 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.
18. A. Sheth, K. Verna, J. Miller, and P. Rajasekaran. EnhacingWeb Service Descriptions usingWSDLS. In *EclipseCon*, 2005.
19. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.
20. P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. ISWC'04*, 2004.
21. K. Verma, A. Mocan, M. Zarembra, A. Sheth, and J. A. Miller. Linking Semantic Web Service Efforts: Integrationg WSMX and METEOR-S. In *Semantic and Dynamic Web Processes (SDWP)*, 2005.
22. SDK WSMO working group. The Web Service Modeling Framework - http://www.wsmo.org/.
23. D.Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-SWeb Services Composition using SHOP2. In *Proc. ISWC'03*, 2003.