Web Services Description Language (WSDL) 2.0 Part 0: Primer

Рекомендація W3C від 26 червня 2007

Ця версія:
http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626
Остання версія:
http://www.w3.org/TR/wsdl20-primer
Попередня версія:
http://www.w3.org/TR/2007/PR-wsdl20-primer-20070523
Редактори:
Девід Бут, співробітник W3C / Hewlett-Packard
Canyang Кевін Лью, SAP Labs

Будь ласка, зверніться до **помилок** в цьому документі, який може включати в себе деякі нормативні виправлення.

Цей документ буде також доступний в цих ненормативних форматах: PDF, PostScript, XML, і звичайний текст.

Див також **переклади.**

Анотація

Цей документ є доповненням до специфікації 2.0 WSDL *(Web Services Description Language (WSDL), версія 2.0 Part 1: основна мова [WSDL 2.0 Core], Web Services Description Language (WSDL) 2.0 Частина 2: добавки [WSDL добавки 2,0])* . Він призначений для читачів, які хотіли б мати більш легке і менш технічне введення в основні особливості мови.

Цей грунт призначений лише для того, щоб стати відправною точкою до використання WSDL 2.0, а не описати всі особливості мови. Користувач, як очікується,може звернутися до WSDL 2.0 специфікації, якщо він хотів би використовувати більш складні функції та методи.

Нарешті, цей грунт не *є нормативним.* Будь-які конкретні питання про те, що WSDL 2.0 вимагає або забороняє слід віднести до WSDL 2.0.

Статус цього документа

Цей розділ описує статус цього документа на момент його публікації. Інші документи можуть замінять цей документ. Список поточних публікацій W3C і остання ревізія цієї технічної доповіді можна ознайомитися в W3C технічного індексу доповідей на http://www.w3.org/TR/.

Це Рекомендація W3C від Web Services Description Language (WSDL) 2.0 Part 0: Primer для розгляду членами W3C та іншими зацікавленими сторонами. Він був

підготовлений за Опис веб-служб робочих груп, який є частиною активності W3C Web Services.

Будь ласка, надсилайте коментарі про цей документ, в суспільні public-ws-desc-comments@w3.org список розсилки (загальнодоступний архів).

Робоча група випустила набір тестів поряд із здійсненням доповіді. Diff-версії відзначені в порівнянні з попередніми версіями цього документа .

Цей документ був розглянутий членами W3C, розробниками програмного забезпечення, а також іншими групами W3C і зацікавленими сторонами, і схвалений Директором в якості рекомендації W3C. Це постійний документ і може бути використаний в якості довідкового матеріалу або цитуватися в інших документах. Роль W3C у розробці даної рекомендації полягає в залученні уваги до цієї специфікації і сприяти її широкому розповсюдженню. Це підвищує функціональність і можливість взаємодії в Інтернеті.

Цей документ регламентується з 24 січня 2002 CPP з поправками, внесеними W3C патентної політики перехідної процедури. W3C підтримує публічний список будь-яких патентів, зроблених у зв'язку з результатами роботи групи; ця сторінка містить інструкції для розкриття патентів. Будь-яка особа, якій фактично було відомо про патентну особу яка, містить основні претензії) повиненна розкрити інформацію згідно з розділом 6 W3C з патентної політики.

Зміст

Програми

---

1. Введення

1.1 Передумови

Цей грунт припускає, що читач має наступні знання передумов:

- знайомство з XML (Extensible Markup Language (XML) 1.0 [XML 1.0], XML Information Set [XML Information Set]) і простором імен XML (Простору імен в XML [простору імен XML]);
- деякий досвід роботи з XML-схемами (XML Schema Part 1: Structures [XML Schema Структури] XML Schema Part 2: Datatypes [типи даних XML Schema]);
- знайомство з основними веб-службами таких понять, як веб-служби, клієнт, а також мета і функції опису веб-служби. (Для роз'яснення основної концепції веб-служби див на веб-послуги Архітектура [Стара архітектура] розділі 1.4 і веб-служб Глосарій [Стара Глосарій] Глосарій. Однак слід зазначити, архітектура Web Services документа використовує більш точні терміни "запит агента" і " постачальник агент "замість термінів" клієнт "і" Веб-сервіс ", які використовуються в цій грунтовці.)

Передбачається, без попереднього досвіду з WSDL.

1.2 Структура цього Primer

Розділ 2 починається з гіпотетичного випадку, пов'язаного з використанням послуг бронювання готелів. Воно відбувається крок за кроком, шляхом створення простого прикладу WSDL 2.0 документ, що описує цю службу:

- `types` елементів описуються *види* повідомлень, що служба буде відправляти і отримувати.
- `interface` елементів описується, *що* анотація функціонально веб-службі надає.
- `binding` елементів описується, *як* отримати доступ до служби.
- `service` елементів описується, *де* доступ до послуги.

Після представлення, наприклад, воно переходить до впровадження WSDL 2.0 Infoset, схеми і моделі компонентів. Потім він забезпечує більш детальне покриття за визначенням типів, інтерфейсів, прив'язок і послуг.

Розділ 3 пояснює, WSDL 2.0 імпорт механізмів у великій деталі.

Розділ 4, йдеться про WSDL 2.0 розширюваність і різні зумовлені розширення.

Розділ 5 охоплює різні теми, які можуть підпадати під сферу WSDL 2.0, але може надати корисну довідкову на передовій практиці посібників, які можуть бути корисні при авторизації WSDL 2.0 документа або здійснення WSDL 2.0.

1.3 Використання URI, і Айріс

Основні специфікації WSDL 2.0 підтримує інтернаціоналізація ідентифікаторів ресурсів або Iris [IETF RFC 3987]. Синтаксис URI [IETF RFC 3986] дозволяє використовувати невеликий набір символів, включаючи верхній і нижній регістр букв англійського алфавіту, цифри і європейських кілька символів. IRIs дозволяє використовувати символи з широкого спектру мов сценаріїв.

Для простоти прикладу протягом усієї цієї грунтовки використовуються тільки URI. Якщо ви зацікавлені в отриманні додаткової інформації про використання IRIS, ви можете читати папери підготовлені активністю W3C інтернаціоналізації.

1.4 Умовні позначення

У цьому документі використовується кілька імен XML, деякі з яких визначаються стандартами, і деякі з них конкретними додатками. Простір імен загального вигляду "http://greath.example.com/ ..." представляють заявку або контекстно-залежну URI, [IETF RFC 3986]. Відзначимо також, що вибір будь-якого префіксу простору імен є довільним і семантично не значний (див. [XML Information Set]).

Після конвенції XML синтаксис резюме в [WSDL 2.0 Core], ця грунтовка використовує неофіційний синтаксис для опису XML граматики WSDL 2.0 документа:

- Синтаксис виглядає як наприклад XML, але значення вказують на тип даних.
- Символи доповнили елементами і атрибутами наступним чином: "?" (0 або 1), "*" (0 або більше), "+" (1 або більше).
- Елементи імені закінчуються на "..." вказують, що елементи атрибутів мають відношення до контексту.

2. WSDL 2.0 Основи

2.1 Початок роботи: приклад Greath Hotel

У цьому розділі представлені основні поняття, що використовуються в WSDL 2.0 через опис гіпотетичних послуг бронювання готелю. Почнемо з простого сценарію, а потім додамо більше вимог до ілюстрації того, як більш розвинуті WSDL 2.0 можливості можуть бути використані.

2.1.1 Приклад сценарію: Greath Hotel Reservation Service

Hotel Greath (вигаданий Готель) знаходиться на віддаленому острові. Він міг покладатися на факс і телефон, щоб забезпечити бронювання номерів. Хоча умови і ціни на Greath кращі ніж те, що пропонують конкуренти, Greath помічає, що його конкурент має більше клієнтів, ніж Greath. Після дослідження, Greath розуміє, що це тому, що конкурент пропонує веб-служби, що дозволяє Travel Systems агент застереження зарезервувати номери безпосередньо через Інтернет. Greath наймає нас створити веб-сервіс бронювання з наступними функціями:

- *CheckAvailability*. Щоб перевірити наявність вільних місць, клієнт повинен вказати дати прибуття, дати виїзду, а також тип номера. Веб-служба поверне

номер (число в доларах США), якщо такий номер доступний. Якщо вхідні дані недійсні, служба повинна повертати помилку. Таким чином, ця служба буде приймати `checkAvailability` повідомлення і повернутати `checkAvailabilityResponse` або `invalidDataFault` повідомлення.

- *MakeReservation.* Щоб зробити замовлення, клієнт повинен вказати ім'я, адресу і інформацію про кредитну картку, і послуга буде повертати номер підтвердження, якщо застереження є успішним. Служба поверне повідомлення про помилку, якщо номер кредитної картки або будь-якої іншої області має неприпустимі дані. Таким чином, ця служба буде приймати `makeReservation` повідомлення і повернутати `makeReservationResponse` або `invalidCreditCardFault` повідомлення.

Ми знаємо, що пізніше ми зможемо створити повну систему, яка підтримує угоди і забезпечення передачі, але спочатку ми будемо здійснювати лише мінімальну функціональність. Справді, для спрощення нашого першого прикладу, ми будемо здійснювати тільки *CheckAvailability* операції.

Наступні кілька розділів будемо крок за кроком у рамках процесу розробки WSDL 2.0 документу, який описує необхідні веб-служби. Однак, для тих, хто не може чекати, щоб побачити повну версію ось, наприклад, WSDL 2.0 документ, який треба провести.

Приклад 2-1. WSDL 2.0 Документ для Greath Web Service (перший приклад)

```
<? XML Version = "1.0" кодування = "UTF-8"?
<Опис
    XMLNS = "http://www.w3.org/ns/wsdl"
    TargetNamespace = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: TNS = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: ghns = "http://greath.example.com/2004/schemas/resSvc"
    XMLNS: wsoap = "http://www.w3.org/ns/wsdl/soap"
    XMLNS: мило = "http://www.w3.org/2003/05/soap-envelope"
    XMLNS: wsdlx = "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    Цей документ описує службу Greath Інтернеті.  Додаткове
    прикладного рівня вимог для використання цієї послуги --
    окрім того, що WSDL 2.0 здатне описати - доступно
    на http://greath.example.com/2004/reservation-documentation.html
  </ Documentation>

  <types>
    <XS: схема
        XMLNS: XS = "http://www.w3.org/2001/XMLSchema"
        TargetNamespace = "http://greath.example.com/2004/schemas/resSvc"
        XMLNS = "http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element name="checkInDate" type="xs:date"/>
          <xs:element name="checkOutDate" type="xs:date"/>
          <xs:element name="roomType" type="xs:string"/>
        </ XS: послідовність>
      </ XS: ComplexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </ XS: Schema>
  </ Типи>

  <interface Name = "reservationInterface">
```

```
    <вини Name = "invalidDataFault"
            Елемент = "ghns: invalidDataError" />

    <ім'я операції = "opCheckAvailability"
            шаблон = "http://www.w3.org/ns/wsdl/in-out"
            стиль = "http://www.w3.org/ns/wsdl/style/iri"
            wsdlx: безпечні = "true">
        <введення messageLabel = "В"
              Елемент = "ghns: checkAvailability" />
        <вихідний messageLabel = "Out"
              Елемент = "ghns: checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </ ОПЕРАЦІЇ>

  </ Interface>

  <обов'язковий Name = "reservationSOAPBinding"
          Інтерфейс = "TNS: reservationInterface"
          Type = "http://www.w3.org/ns/wsdl/soap"
          wsoap: протокол = "http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <вини Ref = "TNS: invalidDataFault"
      wsoap: Код = "мило: Відправник" />

    <операція Ref = "TNS: opCheckAvailability"
      wsoap: MEP = "http://www.w3.org/2003/05/soap/mep/soap-response" />

  </ Binding>

  <ім'я служби = "reservationService"
       Інтерфейс = "TNS: reservationInterface">

     <кінцева точка Name = "reservationEndpoint"
               обов'язковий = "TNS: reservationSOAPBinding"
               Адреса = "http://greath.example.com/2004/reservation" />

  </ Послугу>

</ Description>
```

2.1.2 Визначення WSDL 2.0 Target Namespace

Перед тим, як писати наш WSDL 2.0 документ, ми повинні прийняти рішення про WSDL 2.0 *цільового простору* імен URI для нього. WSDL 2.0 цільових імен аналогічно цільовому простору імен XML Schema. Інтерфейс, імена сервісів, що ми визначимо в нашому WSDL 2.0 документі будуть пов'язані з WSDL 2.0 цільовим простором імен, і, отже, будуть відрізнятися від аналогічного імені в інших WSDL 2,0 цільового простору імен. (Це стане важливим при використанні WSDL 2.0 "з імпортом або інтерфейс механізмами успадкування.)

Значення WSDL 2.0 цільового простору імен повинно бути абсолютним URI. Крім того, воно має бути dereferenceable до WSDL 2.0 документа, який описує веб-службу, WSDL 2.0 цільового простору імен для опису. Наприклад, Greath власники повинні зробити WSDL 2.0 документ який можна завантажити з цієї URI. (А якщо WSDL 2.0 опис розбивається на кілька документів, то WSDL 2.0 цільових імен має вирішити, щоб головний документ, який містить всі WSDL 2.0 документи, мав необхідні для цієї служби описи.) Разом з тим, немає абсолютної необхідності цієї URI для dereferenceable бути, тому WSDL 2.0 процесор не повинен залежати від його часу dereferenceable.

Ця рекомендація може видатися круговою, але майте на увазі, що клієнт міг отримати WSDL 2.0 документ з будь-якого місця - не обов'язково є авторитетним

джерелом. Але WSDL 2.0 цільовий простір імен URI, користувач повинен мати можливість отримати авторитетну версію. З Greath буде власником цієї служби, WSDL 2.0 цільовий простір імен URI слід послатися на місці на веб-сайт Greath або іншим чином в рамках свого контролю.

Після того як ми вирішили на WSDL 2.0 цільовий простір імен URI, ми можемо розпочати наш WSDL 2.0 документ у вигляді наступної порожньої оболонки.

Приклад 2-2. Початковий порожній документ WSDL 2.0

```
<? XML Version = "1.0" кодування = "UTF-8"?
<Опис
    XMLNS = "http://www.w3.org/ns/wsdl"
    TargetNamespace = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: TNS = "http://greath.example.com/2004/wsdl/resSvc"
    .  .  .  >

  .  .  .
</ Description>
```

2.1.2.1 Пояснення прикладу

`<description>`

Кожний WSDL 2.0 Документ description елемента, як найвищий елемент. Це виступає лише як контейнер для іншої частини документа WSDL 2.0, і використовується для оголошення просторів імен, які будуть використовуватися в усьому документі.

`xmlns="http://www.w3.org/ns/wsdl"`

Цей простір імен XML для WSDL 2.0 себе. Ми призначаємо його в якості імен за замовчуванням для цього прикладу, не визначальний префікс відсутній. Іншими словами, будь-який unprefixed елемент в даному прикладі передбачає WSDL 2.0 елементи (такі як `description` елементи).

`targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"`

Це визначає WSDL 2.0 цільовий простір імен, які ми обрали для послуги бронювання Greath, яке описано вище. Зауважимо, що це не є реальним простором імен XML декларації. Скоріше, це WSDL 2.0 атрибут, мета якого є *аналогічною* цільовому простору імен XML-схеми.

`xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"`

Це фактичний простір імен XML декларації для використання в нашому описі послуги Greath. Зауважимо, що це той же URI, який було вказана вище як значення `targetNamespace` атрибутом. Це дозволить нам пізніше використати `tns:` префікс в тих QName, послатися на WSDL 2.0 цільових імен служби Greath. (Детальніше про тих QName див. *[простору імен XML]* розділ 3 Повні імена.)

Тепер ми можемо почати опис послуг Greath.

2.1.3 Визначення типів повідомлень

Ми знаємо, що послуга відправки і отримання повідомлень буде, тому важливим при описі послуг є визначення типів повідомлень, що служба буде використовувати. Ми будемо використовувати XML-схеми, зробимо це, тому що WSDL 2.0 процесори, швидші за все, підтримка XML-схем на мінімальному рівні. Однак, WSDL 2.0 не забороняє використання будь-якої іншої мови визначення схеми.

WSDL 2.0 дозволяє типи повідомлень, які будуть визначені безпосередньо в WSDL 2.0 документі, всередині `types` елемента, який є дочірнім для `description` елементів. (Пізніше ми побачимо, яким чином ми можемо забезпечити тип визначень в

окремому документі, використовуючи `import` механізм XML Schema's.) Визначаємо наступну схему `checkAvailability checkAvailabilityResponse` і `invalidDataError` повідомлень, які нам знадобляться.

У WSDL 2.0, і всі нормальні типи повинні бути визначені як один *елемент* на верхньому рівні (хоча, звичайно, кожен елемент може мати будь-яку кількість підструктури всередині неї). Таким чином, тип повідомлення не повинен безпосередньо складатися з послідовності елементів або іншого складного типу.

Приклад 2-3. Greath типи повідомлень

```
<? XML Version = "1.0" кодування = "UTF-8"?
<Опис
    XMLNS = "http://www.w3.org/ns/wsdl"
    TargetNamespace = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: TNS = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: ghns = "http://greath.example.com/2004/schemas/resSvc"
    .  .  .  >

  ...

  <types>
    <XS: схема
        XMLNS: XS = "http://www.w3.org/2001/XMLSchema"
        TargetNamespace = "http://greath.example.com/2004/schemas/resSvc"
        XMLNS = "http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element name="checkInDate" type="xs:date"/>
          <xs:element name="checkOutDate" type="xs:date"/>
          <xs:element name="roomType" type="xs:string"/>
        </ XS: послідовність>
      </ XS: ComplexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </ XS: Schema>
  </ Типи>
  .  .  .
</ Description>
```

2.1.3.1 Пояснення прикладу

`xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"`

Ми додали інший простір імен декларації. Префікс `ghns` імен дозволить нам (пізніше, при визначенні інтерфейсу) звернутися до цільового простору імен XML-схеми, і ми визначимо наші типи повідомлень. Таким чином, ми вказуємо що визначили мету імен наших типів схем XML (див. нижче) - *не* цільовий простір імен WSDL 2.0 самого документа.

`targetNamespace="http://greath.example.com/2004/schemas/resSvc"`

Це є об'єктом простору імен XML-схеми, яку ми створили для використання послуги бронювання Greath. Імена `checkAvailability checkAvailabilityResponse` і `invalidDataError` елементів будуть асоціюватися з даним цільовим простором імен XML-схеми.

`checkAvailability checkAvailabilityResponse` і `invalidDataError`

Це типи повідомлень, які ми будемо використовувати. Зауважимо, що ці визначаються як *XML-елементи,* як описано вище.

Хоча ми визначили декілька типів, ми ще не вказали, які з них повинні бути використані в якості типів повідомлень на веб-служби. Ми зробимо це в наступному розділі.

2.1.4 Визначення інтерфейсів

WSDL 2.0 дозволяє відокремити абстрактні описи функціональності веб-служб від конкретних деталей, як і пропонують цю функціональність. Такий поділ сприяє різним рівнем повторного використання та розподілу роботи в житті веб-служби і WSDL 2.0 документу, який описує його.

WSDL 2.0 `interface` визначає абстрактний інтерфейс веб-служб як набір абстрактних *операцій,* кожна операція представляє просту взаємодію між клієнтом і послугою. Кожна операція визначає типи повідомлень, які служба може відправляти й одержувати в рамках цієї операції. Кожна операція також визначає *модель* обміну повідомленнями, що свідчить про послідовність, в якій пов'язані повідомлення повинні передаватися між сторонами. Наприклад, *In-Out* характер (див. *WSDL 2.0 Попереднє визначення розширення [WSDL 2.0 придатків]* розділ 2.2.3 In-Out) вказує, що, якщо клієнт посилає повідомлення ***в*** службу, служба відправить відповідь *назад* до клієнта (у звичайному випадку), або він буде посилати повідомлення назад до клієнта (в разі помилки). Ми будемо пояснювати більше повідомлень *про* модель обміну з в **розумінні 2.4.4.3 Message Exchange Patterns (Європарламентарії)**

За послугу Greath, ми (спочатку) визначили інтерфейс, що містить одну операцію, `opCheckAvailability` використовуючи `checkAvailability` і `checkAvailabilityResponse` типи повідомлень, які ми визначили в розділ `types`. Ми використовуємо In-Out шаблон для цієї операції, тому що це найбільш природний спосіб представлення простого запит-відповідь взаємодії. Ми могли б замість (наприклад) визначених двох окремих операцій, використовуючи в тільки і тільки з-моделі (див. *WSDL 2.0 Попереднє визначення розширення [WSDL 2.0 придатків]* розділ 2.2.1 По-Only, а в розділі 2.2.5 З-Only), Але це тільки ускладнить ситуацію для клієнта, тому що нам в такому випадку доведеться окремо вказувати розробник клієнта, що дві операції повинні використовуватися спільно, як "запит-відповідь» пара.

На додаток до нормальних вхідних і вихідних повідомлень, ми також повинні визначити провину послання, яке ми хотіли б використовувати у разі помилки. WSDL 2.0 дозволяє щоб повідомлення про помилки були оголошені в `interface` елементі з метою полегшення повторного використання недоліків всієї операції. Якщо помилка відбувається, коли вона припиняє всі послідовності повідомлень буде вказано на шаблон обміну повідомленнями операції.

Давайте додамо до цих наш документ WSDL 2.0.

Приклад 2-4. Greath Interface Definition

```
<? XML Version = "1.0" кодування = "UTF-8"?
<Опис
    XMLNS = "http://www.w3.org/ns/wsdl"
    TargetNamespace = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: TNS = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: ghns = "http://greath.example.com/2004/schemas/resSvc"
    .  .  .
    XMLNS: wsdlx = "http://www.w3.org/ns/wsdl-extensions">

  .  .  .
  <types>
       ...
  </ Типи>

  <interface Name = "reservationInterface">

    <вини Name = "invalidDataFault"
```

```
            Елемент = "ghns: invalidDataError" />

      <ім'я операції = "opCheckAvailability"
            шаблон = "http://www.w3.org/ns/wsdl/in-out"
            стиль = "http://www.w3.org/ns/wsdl/style/iri"
            wsdlx: безпечні = "true">
          <введення messageLabel = "В"
                Елемент = "ghns: checkAvailability" />
          <вихідний messageLabel = "Out"
                Елемент = "ghns: checkAvailabilityResponse" />
          <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
      </ ОПЕРАЦІЇ>

    </ Interface>

    .  .  .
  </ Description>
```

2.1.4.1 Пояснення прикладу

<interface name = "reservationInterface" >

Інтерфейси предявленні безпосередньо в `description` елементі. У цьому прикладі ми використовуємо тільки один інтерфейс, але в цілому 2.0 WSDL документ може використовувати більш ніж один інтерфейс. Таким чином, кожен інтерфейс повинен мати ім'я, яке є унікальним в межах встановлених інтерфейсів, визначених у даному WSDL 2.0 цільового простору імен. Інтерфейс імені не повинен містити пробіл або двокрапка (":").

<fault name = "invalidDataFault"

`name` атрибут визначає ім'я. Назва необхідна для того, що, коли операція визначена, вона може посилатися на бажані імена. Несправність імені повинна бути унікальною в межах інтерфейсу.

<element = "ghns:invalidDataError"/>

`element` атрибут визначає тип схеми, з помилки повідомлення, як було визначено в розділі `types`

<operation name="opCheckAvailability"

`name` атрибут визначає ім'я для цієї операції, на яке можна посилатися пізніше, коли вже визначені прив'язки. Операція імені повинна бути унікальною в межах інтерфейсу. (WSDL 2.0 використовує окремий символом простору для роботи і усунення несправностей імені, тому ім'я операції "Foo" відрізняється від помилки назвою "Foo".)

«pattern="http://www.w3.org/ns/wsdl/in-out»

Цей рядок визначає, що ця операція буде використовуватися In-Out схемою, описаною вище. WSDL 2.0 URI, використовується для визначення моделей обміну повідомленнями з метою забезпечення того, щоб ідентифікаторів глобально ознайомити, а також дозвіл на майбутні нові моделі, які будуть визначені пізніше. (Втім, тільки тому, що хтось визначає новий шаблон і створює URI для його ідентифікації, це *не означає,* що інші WSDL 2.0 процесори будуть автоматично розпізнавати і розуміти цю схему. Як і будь-яке інше розширення, воно може бути використано тільки між процесорами, щоб *робити*, визнавати і розуміти його.)

«style="http://www.w3.org/ns/wsdl/style/iri»

Цей рядок показує, що схема XML визначення вхідного повідомлення цієї операції наступного набору правил, як зазначено в стилі IPI, яке забезпечувало повідомлення може бути представлено як серія «IPI».

<wsdlx:safe="true">

Цей рядок показує, що ця операція не зобов'язує клієнта, тобто клієнт може безпечно викликати цю операцію без страху, і не нести зобов'язання (наприклад, згода на щось купити). Це пояснюється в **2.4.4 Інтерфейс операціях.**

<input messageLabel="In" >

`input` елемент визначає введення повідомлення. Хоча ми вже вказували, які моделі обміну повідомленнями операція буде використовувати шаблон обміну повідомленнями являє собою шаблон для послідовності повідомлень, і в теорії могла б складатися з декількох вхідних та / або вихідних повідомлень. Таким чином, ми повинні також вказати, які потенційні повідомлення введенні в структуру даного повідомлення. Це мета `messageLabel` атрибута. Після відправлення в шаблон, який ми обрали для використання лише одного вхідного повідомлення, ми просто заповнимо повідомленні етикеткою "В", яка була визначена у *WSDL 2.0 Попереднє визначення розширення [WSDL 2.0 придатків]* розділ 2.2.3 In-Out на In-Out шаблоном. Однак, якщо нова модель визначена, що включають декілька повідомлень введення, то різні повідомлення введення в шаблоні можна було б відрізняти використовуючи різні ярлики.

<element="ghns:checkAvailability" />

Це визначає тип повідомлення вхідного сигналу, як визначено раніше в розділі `types`

<output messageLabel="Out" . . .

Це схоже на визначення вхідних повідомлень.

<outfault ref="tns:invalidDataFault" messageLabel="Out"/>

Це пов'язує вихід з цією операцією. Це оголосили дещо інакше, ніж звичайні повідомлення. `Ref` атрибути відноситься до назви раніше визначеної провини в цьому інтерфейсі - не тип повідомлення схеми безпосередньо. Оскільки структура обміну повідомленнями може включати в загальному послідовність з декількох повідомлень, то це потенційно може відбуватися в різних точках в послідовності повідомлень. Використовується `messageLabel,` вказує потрібну точку для даної конкретної помилки. Він робить це побічно, вказавши, що повідомлення будуть викликати цей недолік або що цей недолік буде замінений, залежно від моделі. (Деякі моделі використовують повідомлення тригерів за замовчуванням правила, інші використовують замінні повідомлення правил. *WSDL 2.0 Попереднє визначення розширення [WSDL 2.0 придатків]* розділ 2.1.2 повідомлення ініціює несправность, в розділі 2.1.1 Несправність замінює повідомлення.)

Тепер, коли ми визначили абстрактний інтерфейс для служби Greath, ми готові визначити для нього обов'язкові.

2.1.5 Визначення Binding

Хоча ми й вказали, *що* абстрактні повідомлення можуть бути обмінені в службі Greath Інтернет, ми ще не вказали, *яким чином* ці повідомлення можуть бути обмінені. Це є *обов'язковою метою.* Обов'язково визначається конкретний формат повідомлення і подробиці протоколу передачі для інтерфейсу, і повиненна надаватися така інформація для кожної операції, і помилок в інтерфейсі.

У загальному випадку, обов'язкові деталі для кожної операції та помилки задаються за допомогою `operation` і `fault` елементів всередині `binding` елементів, як показано в наступному прикладі. Однак у деяких випадках можна використовувати правила для постачання інформації. WSDL, SOAP 2.0 обов'язкові розширення, наприклад, визначає деякі правила для проведення операцій. (Див. *Web Services Description*

Language (WSDL) 2.0 Частина 2: добавки [WSDL добавки 2,0], за замовчуванням обов'язкових правил.)

Для того, щоб пристосувати їх до нових видів форматів повідомлень і протоколів передачі, визначені прив'язки за допомогою розширень для мови WSDL 2.0, через WSDL 2.0 "з відкритою моделлю змісту. (Див. **4.1 Розширюваність** більш докладно про розширюваність.) WSDL 2.0 Частина 2 *[WSDL 2.0 придатків]* визначає обов'язкове розширення для SOAP 1.2 *[SOAP 1.2 Part 1: Messaging Framework]* і HTTP 1.1 *[IETF RFC 2616],* зумовлені як розширення, так що SOAP 1.2 або HTTP 1.1 прив'язки можуть бути легко визначені в WSDL 2.0 документі. Разом з іншими специфікаціями можна визначити нове обов'язкове розширення, яке також може бути використане для визначення прив'язок. (Як і з будь-яким розширенням, інші WSDL 2.0 процесори б дізналися про нові конструкції, з тим щоб використовувати їх.)

За послугу Greath, ми будемо використовувати SOAP 1.2 як наш конкретний формат повідомлення і HTTP як наш основний протокол передачі даних, як показано нижче.

Приклад 2-5. Greath Binding Визначення

```
<? XML Version = "1.0" кодування = "UTF-8"?
<Опис
    XMLNS = "http://www.w3.org/ns/wsdl"
    TargetNamespace = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: TNS = "http://greath.example.com/2004/wsdl/resSvc"
    XMLNS: ghns = "http://greath.example.com/2004/schemas/resSvc"
    XMLNS: wsoap = "http://www.w3.org/ns/wsdl/soap"
    XMLNS: мило = "http://www.w3.org/2003/05/soap-envelope">
  .  .  .

  <types>
    .  .  .
  </ Типи>

  <interface Name = "reservationInterface">
        ...
  </ Interface>

  <обов'язковий Name = "reservationSOAPBinding"
          Інтерфейс = "TNS: reservationInterface"
          Type = "http://www.w3.org/ns/wsdl/soap"
          wsoap: протокол = "http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <операція Ref = "TNS: opCheckAvailability"
      wsoap: MEP = "http://www.w3.org/2003/05/soap/mep/soap-response" />

    <вини Ref = "TNS: invalidDataFault"
      wsoap: Код = "мило: Відправник" />

  </ Binding>

  .  .  .
</ Description>
```

2.1.5.1 Пояснення прикладу

`xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"`

Ми додали ще дві заяви імен. Це один простір імен для SOAP 1.2 обов'язкового розширення, яке визначено в WSDL 2.0 Частина 3 *[SOAP 1.2 Part 1: Messaging Framework].* Елементи й атрибути з префіксом `wsoap:` є конструкціями у даному регіоні.

`xmlns:soap=http://www.w3.org/2003/05/soap-envelope`

Цей простір імен визначається SOAP 1.2 Specification . SOAP 1.2 специфікація визначає деякі терміни, в рамках цього простору імен, щоб однозначно

визначити конкретні концепції. Таким чином, ми будемо використовувати префікс, коли нам потрібно звернутися до одного з цих термінів.

<binding name="reservationSOAPBinding"

Bindings оголошується безпосередньо в описі елемента. Ім'я атрибута визначає ім'я для цього. Кожне ім'я має бути унікальним серед всіх прив'язок в цьому WSDL 2.0 цільового простору імен, і будуть використані в подальшому, коли ми визначимо, що кінцева точка служби посилається на цю силу. WSDL 2.0 використовує окремі символи для інтерфейсів, прив'язки і послуг, тому інтерфейс "Foo", обов'язковий "Foo" і сервіс "Foo" всі різні.

interface="tns:reservationInterface"

Це ім'я інтерфейсу, формат повідомлення та протоколи передачі ми визначаємо. Як зазначалося в **2.5 Детальніше про Bindings**, багаторазовий обов'язковий елемент може бути визначений за допомогою виключення інтерфейс атрибуту. Відзначимо також використання `tns:` префікс, який посилається на раніше визначені WSDL 2.0 цільові простори імен для цього документа WSDL 2.0. У цьому випадку вона може здатися неправильною щоб вказати `tns`: префікс, але в **3.1 Імпорт WSDL** ми побачимо, як WSDL 2.0 'S механізм імпорту може бути використаний для об'єднання компонентів, які визначені в різних WSDL 2.0 імен.

type = "http://www.w3.org/ns/wsdl/soap"

Це вказує, які конкретно формати повідомлень використовуються, в даному випадку SOAP 1.2.

wsoap: протокол = "http://www.w3.org/2003/05/soap/bindings/HTTP/"

Цей атрибут специфічних для WSDL 2.0 "з обов'язковим розширенням SOAP (таким чином, він використовує `wsoap:` префікс). Він визначає основні протоколи передачі даних, які повинні бути використані, в даному випадку HTTP.

<операція ref = "tns: opCheckAvailability"

Це не визначення нової операції; скоріше, це посилання на певні раніше opCheckAvailability операції, з тим щоб вказати обов'язкові для нього подробиці. Цей елемент може бути опущений, якщо правило не використовується для подачі необхідної інформації. (Див. SOAP обов'язкові розширення WSDL 2.0 Частина 2 [*WSDL 2.0 придатків*] розділ 4.3 за замовчуванням обов'язкових правил.)

wsoap: mep = "http://www.w3.org/2003/05/soap/mep/soap-response">

Цей атрибут також специфічний для WSDL 2.0 'S SOAP є обов'язковим розширенням. Він визначає обмін SOAP Message шаблонів (МВП), які будуть використовуватися для реалізації абстрактного WSDL 2.0 обміну повідомлень та шаблонів (in-out), які були зазначені, коли була визначена `opCheckAvailability` операція.

Коли HTTP використовується в якості основного транспортного протоколу (як в даному прикладі) `wsoap:mep` атрибут також контролює використання в якості основного методу HTTP. У цьому випадку, використання `wsoap: mep = "http://www.w3.org/2003/05/soap/mep/soap response"` причина GET, що буде використовуватися за замовчуванням. Див також **2.5.7 HTTP GET Versus POST: Яку використовувати? .**

<faulf ref = "tns: invalidDataFault">

Як і у випадку обов'язкової операції, це не оголошення нових проблем; скоріше, це посилання причини (`invalidDataFault`), яка була раніше визначена у `opCheckAvailability` інтерфейсі, для того, щоб вказати обов'язкові для нього подробиці.

<wsoap: code = "soap: Відправник" />

Цей атрибут також специфічний для WSDL 2.0 'S SOAP обов'язкового розширення. Це визначає SOAP 1.2, який викличе цей недолік і буде відправлено повідомлення. При бажанні, список також може бути зазначений з використанням додаткового `wsoap:`атрибуту.

2.1.6 Визначення служби

Тепер, *які* повідомлення будуть передаватися, ми готові вказати, *де* послуги можна отримати, шляхом використання `службового` елемента.
WSDL 2.0 Service визначає єдиний інтерфейс,який служба буде надавати на підтримку, а також список кінцевих місць, де ця служба може бути доступною. Кожна кінцева точка повинна також посилатися на раніше визначені обов'язкові, а також які протоколи і формати передачі будуть використовуватися на цій кінцевій точці. Служба дозволяє тільки один інтерфейс. (Див. **5.4 Множинні інтерфейси для тієї ж служби** для подальшого обговорення цього обмеження.)
Ось визначення для нашого сервісу Greath.
Приклад 2-6. Greath Service Definition

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  . . .

  <types>
    . . .
  </types>

  <interface  name = "reservationInterface" >
    . . .
  </interface>

  <binding name="reservationSOAPBinding"
          interface="tns:reservationInterface"
        . . . >
    . . .
  </binding>

  <service name="reservationService"
       interface="tns:reservationInterface">

     <endpoint name="reservationEndpoint"
               binding="tns:reservationSOAPBinding"
               address ="http://greath.example.com/2004/reservation"/>

  </service>

</description>
```

2.1.6.1 Пояснення прикладу

`<ім'я служби = "reservationService"`

Це визначає ім'я для цієї служби, яка повинна бути унікальною серед служб імен в WSDL 2.0 цільового простору імен. Потрібно ім'я атрибута. Це дозволяють URI компоненти в WSDL 2.0 опис. (*Див. WSDL 2.0 Core мови* [WSDL 2.0 Core] додавання 3 IPI Документи WSDL 2.0 конструкцій.)

`<Інтерфейс = "tns: reservationInterface">`

Тут вказується ім'я раніше визначеного інтерфейсу, які ця послуга далі буде підтримувати.

`<кінцева точка name = "reservationEndpoint">`

Це і визначає кінцеву точку для служби, і ім'я цього порогу, який повинен бути унікальним в межах даної послуги.

<обов'язковий = "tns: reservationSOAPBinding" >

Тут вказується ім'я раніше визначеного обов'язкового елемента для використання цієї кінцевої точки.

<адреса = "http://greath.example.com/2004/reservation" />

Це визначає фізичний адрес, за яким ця послуга може бути доступна через обов'язкові передбачені атрибути.

От і все! Ну, майже.

2.1.7 Документування служби

Як ми бачили, WSDL 2.0 документ за своєю суттю лише частково описує послуги. Хоча він відображає основні механіки взаємодії зі службою - типи повідомлень, протоколів передачі, обслуговування місць і т. д. - загалом, додаткову документацію доведеться пояснити іншими додатками рівня вимог до його використання. Наприклад, така документація повинна пояснити мету і використання послуг, сенс всіх повідомлень, обмежень на їх використання, і в якій послідовності операцій слід посилатися.

`Документація` елемент дозволяє WSDL 2.0 включити деякі людського розуміння документації всередині документа WSDL 2.0. Він також є зручним місцем для посилання на додаткову зовнішню документацію, яка клієнт-розробнику може знадобитися для того, щоб скористатися цією службою. Вона може з'являтися в кількох місцях у WSDL 2.0 документі (див. розділ **2.2.1 WSDL 2.0 Infoset**), хоча в даному прикладі ми маємо лише продемонструвати її використання на самому початку.

Приклад 2-7. Документування Greath служби

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    . . . >

  <documentation>
    This document describes the GreatH Web service.  Additional
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
          at http://greath.example.com/2004/reservation-documentation.html
  </documentation>
  . . .
</description>
```

2.1.7.1 Пояснення приклад

<documentation>

Цей елемент є необов'язковим, але гарна ідея його включити. Він може містити довільну інформацію змішаного вмісту.

на http://greath.example.com/2004/reservation-documentation.html

Саме головне, щоб включити вказівник на будь-яку додаткову документацію, розробник- клієнту буде потрібно, щоб скористатися цією службою.

Це завершує нашу презентацію на прикладі Greath. У наступних розділах ми будемо рухатися більш докладно про різні аспекти WSDL 2.0.

2.2 WSDL 2.0 Infoset, схеми і моделі компонентів

У комп'ютерній теорії науки, мова складається з (можливо, нескінченної) безлічі

пропозицій, і кожна пропозиція являє собою кінцевий рядок літерних символів або просто символів. Мова специфікації визначає безліч пропозицій на цій мові, і, щоб бути корисними, вони повинні також вказати значення кожної пропозиції. Дійсно, це мета WSDL 2.0.

Однак, замість визначення WSDL 2.0 з точки зору буквального символ, щоб уникнути залежності від якого-небудь конкретного кодування, WSDL 2.0 визначається з точки зору *XML Infoset* [*XML Information Set*]. Зокрема, WSDL 2.0 *документ* складається з `інформаційного` елемента опису елемента (у XML Infoset), що відповідає WSDL 2.0. Іншими словами, покарання у WSDL 2.0 мови опису `інформації` елементом, який підпорядковує додаткові обмеження, викладені в WSDL 2.0.

Оскільки XML Infoset може бути створений з більш ніж одного фізичного документа, WSDL 2.0 документ не обов'язково відповідає одному фізичному документу: слово "документ" вживається в переносному сенсі, для зручності. Крім того, оскільки WSDL 2.0 дозволяє імпортувати і включати в себе механізми, WSDL 2.0 документ може містити посилання на інші WSDL 2.0 документи, щоб полегшити використання. У таких випадках, сенс в тому числі і при імпорті документа в цілому буде залежати "(частково) від значення включення або імпортованого документу.

XML Infoset використовує такі терміни, як "елемент пункту інформації" та "атрибуту елемента інформації". На жаль, ці умови є досить тривалими, повторюються часто. Таким чином, для зручності, ця грунтовка часто використовуються терміни "елемент" і "атрибут". Слід розуміти що, оскільки WSDL 2.0 заснований на XML Infoset, ми насправді пояснюємо "пункт елемента інформації" та "атрибуту елемента інформації", відповідно.

2.2.1 WSDL 2.0 Infoset

Така діаграма дає загальне уявлення про XML Infoset для WSDL 2.0 документа

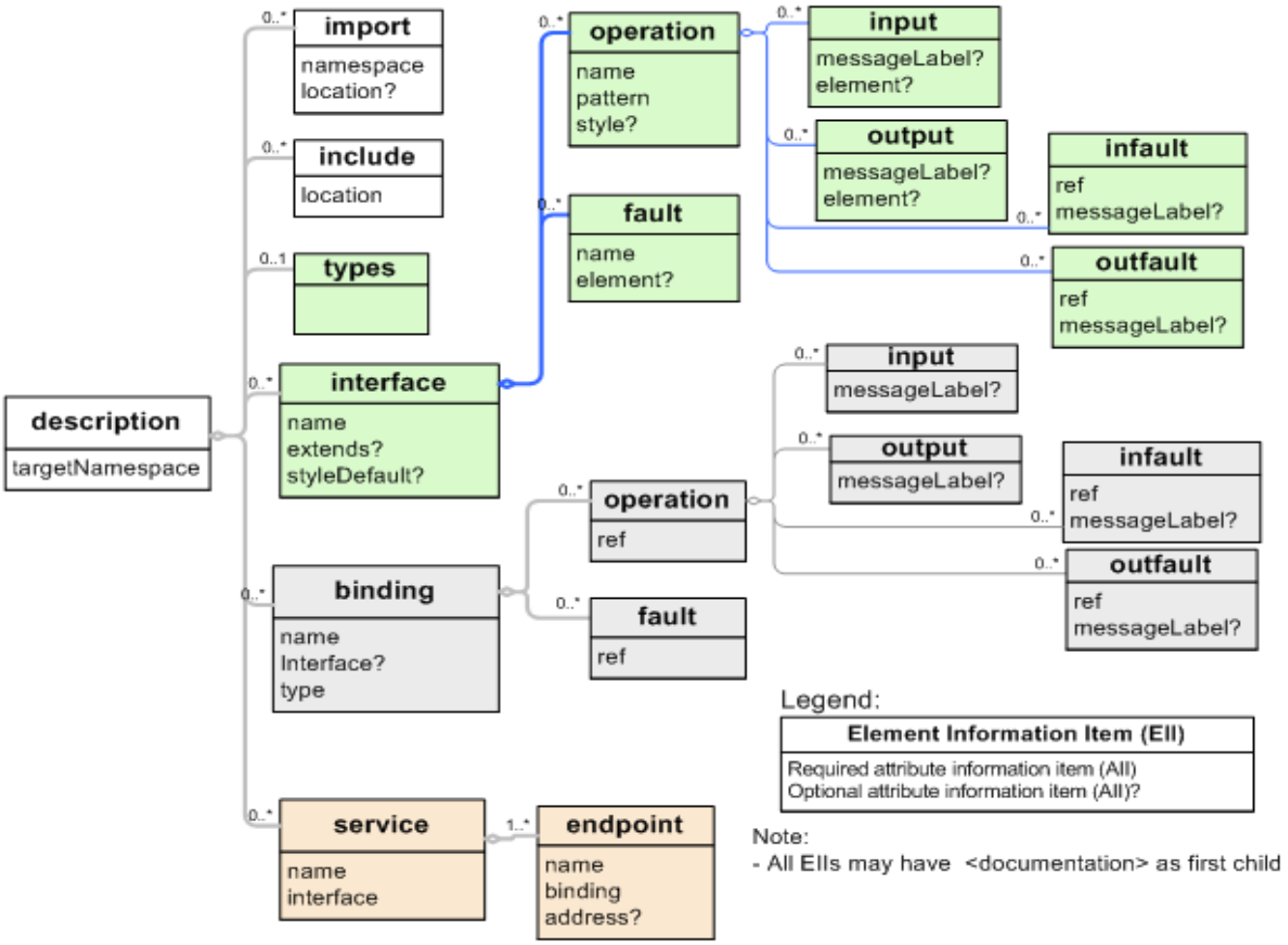

Figure 2-1. WSDL 2.0 Infoset Diagram

2.2.2 WSDL 2.0 Schema

WSDL 2.0 специфікація поставки нормативної WSDL 2.0 схеми, визначеної в [*XML Schema Структури*], яка може бути використана в якості допомоги в WSDL 2.0 перевірки документів. Ми говоримо "в якості допомоги", тут, тому що WSDL 2.0 Specification [*WSDL 2.0 Core*] часто надає додаткові обмеження на WSDL 2.0 схеми. На додаток до того, документ повинен також стежити за усіма обмеженнями визначення WSDL 2.0.

2.2.2.1 WSDL 2.0 Елементу замовлення

Цей розділ дає приклад того, як WSDL 2.0 специфікація обмежує WSDL 2.0 схемою про впорядкування Top WSDL 2.0 елементів.
Хоча WSDL 2.0 схемі не вказані необхідні упорядкування елементів, WSDL 2.0 Specification (WSDL 2.0 частина 1 [*WSDL 2.0 Core*] розділ "XML поданні Опис компонентів") чітко вказується набір обмежень, про те, як дочірні елементи опису елементів повинні бути замовлені. Таким чином, схема не враховує це обмеження. Нижче наводиться зміст псевдо-моделі `опису`.

```
<description>
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

Іншими словами, елементи `опису` елемента повинні бути відсортовані в такий спосіб:
• Факультативна `документація` поставляється по-перше, якщо вона присутня.
• Потім йде нуль або більше елементів з числа наступних, в будь-якому порядку:
- `Про включення`
- `Про імпорт`
- `Про розширення`

•Наступні `типи факультативні`
• Нуль або більше елементів з числа наступних, в будь-якому порядку:
- `інтерфейс`
- `Про обов'язок`
- `Про службу`
- `Про розширеннях`.

Зверніть увагу на термін "розширення" використовується насамперед як зручний спосіб звернутися до простору імен кваліфікованого елемента розширення. Простори імен таких елементів, розширення не повинно бути "http://www.w3.org/ns/wsdl".

2.2.3 WSDL 2.0 Компонентна модель

WSDL 2.0 Infoset вище модель ілюструє структуру необхідних документів WSDL 2.0, з використанням XML Infoset. Тим не менше, WSDL 2.0 мова також накладає багато семантичних обмежень над структурною відповідністю цим XML Infoset. Для того, щоб точно описати ці обмеження, а також допомогти в точному визначенні значення кожного документа WSDL 2.0, WSDL 2.0 специфікація визначає *компонент моделі* в якості додаткового рівня абстракції над XML Infoset. Обмеження і значення визначаються в термінах цієї моделі компонентів, а також визначення кожного компонента включає в себе відображення, яке визначає, які значення в компонентній

моделі випливають з відповідних пунктів в XML Infoset. Така діаграма дає загальне уявлення про WSDL 2.0 компоненти та їх стримування ієрархії.

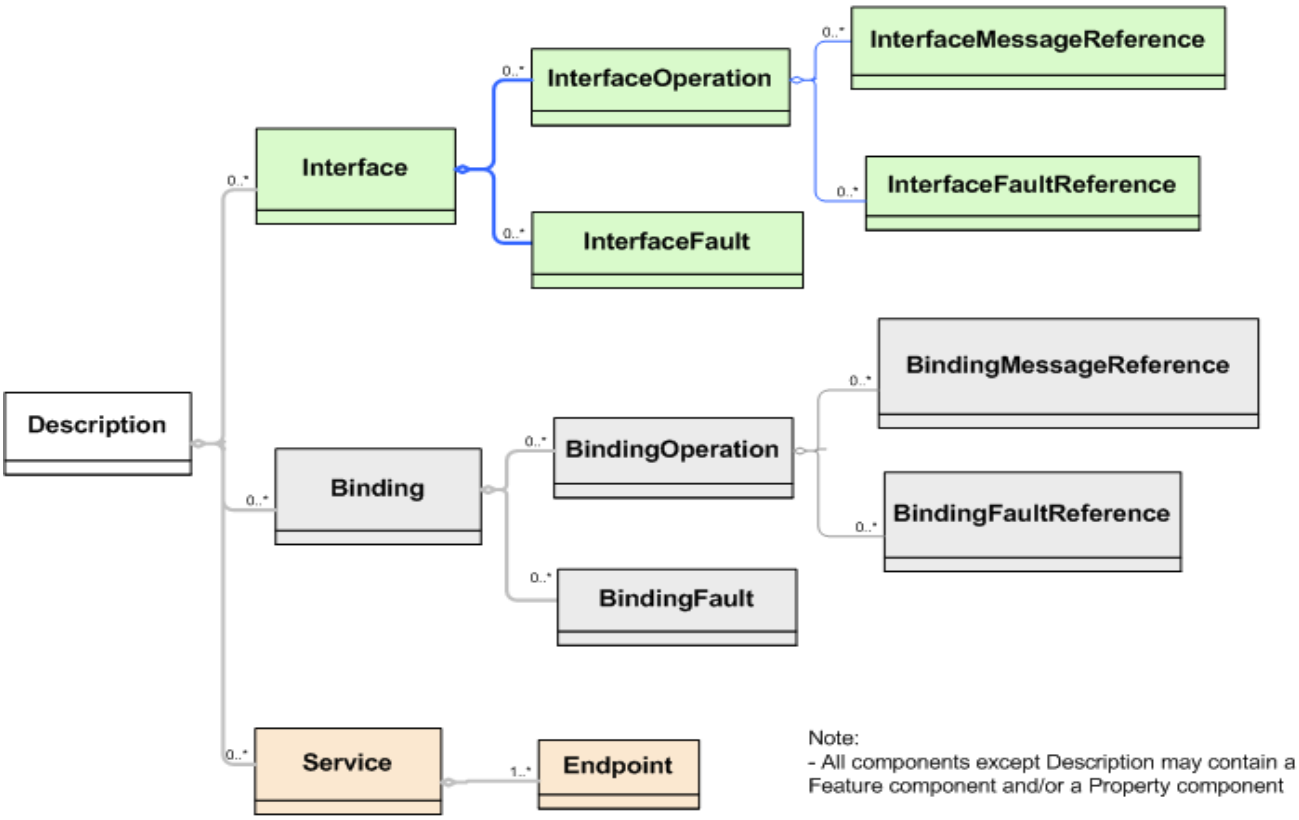

Figure 2-2. WSDL 2.0 Components Containment hierarchy

Загалом, WSDL 2.0 компонентної моделі parallels структура необхідної XML Infoset показано вище. Наприклад, *описи інтерфейсів, палітурні роботи, служби та кінцеві компоненти* відповідають опису, інтерфейсу, палітурній роботі, службі і кінцевим пунктам елементів інформації, відповідно. Так як WSDL 2.0 спирається на компонентні моделі для передачі змісту конструкції в WSDL 2.0 мовою, ви можете думати про опис компонент, що відображає зміст інформації представляє елемент опису, і, отже, вона являє собою WSDL 2,0 документ у цілому.
Крім того, кожен з цих компонентів має властивості, значення яких (як правило), отримані з атрибутів і елементів інформаційного елемента з тих пунктів елементів інформації. Наприклад, компонент служби відповідний інформаційний елемент служби, представляє собою набір компонентів, відповідних Endpoint до кінцевої точки елемента інформації, що інформація пункту елемента сервісу. (Уф!)

2.2.3.1 WSDL 2.0 Імпорт і Увімкнути

WSDL 2.0 компонентна модель особливо корисна при визначенні сенсу імпортувати і включати елементи. Елемент включає дозвіл зібрати вміст даного WSDL 2.0 з декількох імен WSDL 2.0 документів, що визначають компоненти для цього простору імен. Компоненти, що визначаються WSDL 2.0 даного документа складається з тих, визначення яких містяться в документі, і ті, які визначаються будь-якими WSDL 2.0

документами, які включені до неї за допомогою елемента `включають`. Ефект елемента включають є кумулятивним, так що якщо документ включає В документ і документ Б `включає` документ С, то компоненти, визначені в документі складаються з тих, визначення яких містяться в документах, В і С.
На відміну від імпорту елемента не визначає будь-які компоненти. Замість цього, імпорт елементу заявляє, що компоненти, визначення яких містяться в документі WSDL 2.0 для даного WSDL 2.0 імен відносяться до компонентів, які належать до різних WSDL 2.0 імен. Якщо WSDL 2.0 Документ містить визначення компонентів, які посилаються на інші імена, то ці імена повинні бути оголошені через імпорт елементів. Імпорт елементів також має додатковий атрибут місця, що є натяком на процесор, де визначення імпортованих імен може бути знайдене. Однак, процесор може знайти визначення за допомогою інших засобів, наприклад, за допомогою каталогу.
Після обробки включає в себе елементи і розміщення компонентів, які відносяться до будь-якої продукції, що імпортується, WSDL 2.0 компонентної моделі для WSDL 2.0 Документ буде містити набір компонентів, які відносяться до WSDL документа. Ці компоненти будуть називати один одного, як правило, через QName посилання. WSDL 2.0 документ є недійсним, якщо будь-якого компоненту відліку не може бути вирішено, чи посилання компонентів відноситься до того ж або іншого імені.
Ми будемо охоплювати набагато більше про те, як використовувати WSDL 2.0 імпорту і включення в **3.1 Імпорт WSDL**

2.3 More on Message Types

Типи повідомлень можуть бути включенні в різні Мови схемою. Тут ми будемо акцентувати увагу тільки на використання XML Schema [*XML Schema Структури*], оскільки це спочатку підтримується WSDL 2.0. Типи повідомлень, визначені в інших мовах можуть бути введені в WSDL 2.0 Опис за допомогою розширень W3C відзначає [*Альтернативна схема Мови Підтримки*] для більш докладної інформації.
Нижче наводиться синтаксис XML для WSDL: типи елементів:

```
<description>
  < types >
    <documentation />*
    [ <xs:import namespace=" xs:anyURI " schemaLocation=" xs:anyURI "? /> |
      <xs:schema targetNamespace=" xs:anyURI " /> |
      other extension elements  ]*
  </ types >
</description>
```

Є два способи зробити визначення схем XML повідомлення або іншими словами, доступні для ознайомлення за QName (див. WSDL 2.0 частина 1 [*WSDL 2.0 Core*] "QName резолюцію") у WSDL 2.0 документа: вбудовуванні або імпортера.
Вбудовування це помістить схему визначень безпосередньо в `xs: схема` елемента під `типів`. Імпорт виробів проводиться за схемою визначеною в окремому документі, а потім привести його у визначення WSDL за допомогою `xs: імпорт` безпосередньо під `типів`.
У наступних розділах ми будемо служити прикладом для різних механізмів.

2.3.1 Вбудовування XML Schema

Ми вже бачили приклад використання вбудованих визначень схем в розділі **2.1.3 Визначення типів повідомлень**. Коли XML Schema є вбудованою безпосередньо в документ WSDL 2.0, вона використовує існуючу на вищому рівні `xs: схему` елемента, визначається XML-схема, щоб зробити так, як ніби файл схеми був скопійований і вставлений в елемент `типу`. Схема компоненти потім стає доступною для WSDL 2.0

містить опис посилання по QName. Наприклад, в прикладі 2-1, щоб написати в інтерфейсі операцію "opCheckAvailability" визначається за "ghns: checkAvailability" елемента в вбудованих схем.

2.3.2 Імпорт XML Schema

XML Schema компоненти може бути визначена в окремому файлі схеми і бути доступною для WSDL2.0 опису за допомогою xs: імпорт безпосередньо під типів.
Однією з причин є можливості багаторазового використання схем визначення. Вбудовані визначення схеми доступні тільки містять опис WSDL 2.0. Хоча WSDL 2.0 забезпечує wsdl: механізм імпорту інших файлів WSDL, визначення схеми вбудованої в імпортований документ WSDL не будуть автоматично надаватися Імпорт WSDL 2.0 документу, навіть якщо інші WSDL 2.0 компоненти (таких як інтерфейси, кріплення і т.д.) робити стануть доступні. Тому, якщо хочеться поділитися схемою на кілька визначень WSDL 2.0 з описом, ці визначення схеми замість цього слід помістити в окремі документи XML схеми і імпортувати в кожен WSDL 2.0 опис використовуючи xs: імпорт безпосередньо під типів.
Давайте подивимося на прикладі. Файл з ім'ям повідомлень у Прикладі 2-3 визначені в окремій схемі "http://greath.example.com/2004/schemas/resSvc.xsd" з цільовим іменем "http://greath.example.com / 2004/schemas/resSvc ", визначення схеми можуть бути включені в WSDL 2.0 опис використовуючи XS: імпорт. Пам'ятайте, що тільки з компонентами імпортованих імен "http://greath.example.com/2004/schemas/resSvc" можна ознайомитися в документі WSDL 2.0.

Example 2-8. xs:import ed *Message Definitions that Are Visible to the Containing WSDL 2.0 Description*

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
 .  .  .  >
 .  .  .

<types>
        <xs:import namespace="http://greath.example.com/2004/schemas/resSvc"
                schemaLocation=
"http://greath.example.com/2004/schemas/resSvc.xsd"/>
</types>

 .  .  .
</description>
```

Важливо відзначити, що xs: імпорт використовується безпосередньо під wsdl: типи приділяється різній видимості, ніж xs: імпорт, що використовуються всередині вбудованих схем. Вбудована схема може використовувати різну схему XML XS: імпорт залучає зовнішнє визначення схеми, яке знаходиться в різних просторах імен, однак, хоча ця схема імпортерів механізмом рекомендовані для WSDL 1.1 в WS-I Basic Profile, відповідно до специфікації XML Schema.
Якщо ми змінимо Приклад 2-8 використовувати XML Schema рідному xs: імпорт елементів вбудованих схем, схем компоненти, визначені в просторі імен http://greath.example.com/2004/schemas/resSvc не доступні для нашого прикладу WSDL 2.0 визначення небуде.

Example 2-9. `xs:import` ed Message Definitions in Inlined Schema Are Not Visible to the Containing WSDL 2.0 Description

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"
 targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
 xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
 xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  .  .  .  >
  .  .  .

 <types>
         <xs:schema
targetNamespace="http://greath.example.com/2004/schemas/resSvcWrapper">
                 <xs:import
namespace="http://greath.example.com/2004/schemas/resSvc"
                 schemaLocation=
"http://greath.example.com/2004/schemas/resSvc.xsd"/>
         </xs:schema>
 </types>

  .  .  .
 </description>
```

Звичайно, вбудована схема XML може використовувати XML Schema `xs:включати` елементи відноситься до схем визначення в окремих файлах, коли включена схема не має імен або ж простору імен, включаючи схему. У цьому випадку, відповідно до XML-схеми, включаючя такі компоненти, схема стала частиною включати схему, як ніби вони були скопійовані. Таким чином, схема включає такі компоненти, також доступні для WSDL 2.0 `містять опис` посилання по QName.
Наступний приклад має той же ефект, що і [приклад 2-3:]

Example 2-10. `xs:included` Message Definitions in Inlined Schema Are Visible to the Containing WSDL 2.0 Description

```
<?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"
 targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
 xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
 xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  .  .  .  >
  .  .  .

 <types>
         <xs:schema
targetNamespace="http://greath.example.com/2004/schemas/resSvc">
                 <xs:include schemaLocation=
"http://greath.example.com/2004/schemas/resSvc.xsd"/>
         </xs:schema>
 </types>


 . . .
 </description>
```

2.3.3 Summary of Import and Include Mechanisms

До цих пір ми вже коротко охопили як WSDL імпорту включає схеми. У наступній таблиці наведено подібності і відмінності між WSDL 2.0 і XML Schema та імпорт включення механізмів. Ми будемо багато говорити про ввезення механізмів в 3.1 Імпорт WSDL і 3.2 Імпорт схеми

Table 2-1. Summary of Import and Include Mechanisms

Mechanism	Об'єкт	Значення	Visibility of Schema Components
wsdl:import	WSDL 2.0 Namespace	Declare that WSDL 2.0 components refer to WSDL 2.0 components from a DIFFERENT targetNamespace.	XML Schema Components in the imported Description component are NOT visible to the containing `description` .
wsdl:include	WSDL 2.0 Document	Merge Interface, Binding and Service components from another WSDL 2.0 document that has the SAME targetNamespace.	XML Schema components in the included Description component's { element declarations } and { type definitions } properties are visible to the containing `description` .
wsdl:types/ xs:import	XML Schema Namespace	Declare that XML Schema components refer to XML Schema components from a DIFFERENT targetNamespace.	XML Schema components in the imported namespace are visible to the containing `description` .
wsdl:types/ xs:schema/xs:import	XML Schema Namespace	Declare that XML Schema components refer to XML Schema components from a DIFFERENT targetNamespace.	XML Schema components in the imported namespace are NOT visible to the containing `description` .
wsdl:types/ xs:schema/xs:include	XML Schema Document	Merge XML Schema components from another XML Schema document that has the SAME or NO targetNamespace.	XML Schema components in the included document are visible to the containing `description` .

2.4 Докладніше на кордонах

Раніше ми вже згадали, що WSDL 2.0 являє собою набір операцій. Тим не менш, є деякі додаткові можливості, які ми ще не охопили. По-перше, давайте розглянемо синтаксис елементу інтерфейсу.

2.4.1 Інтерфейс Синтаксис

Нижче наводиться резюме XML синтаксис елементу `інтерфейсу`, пропустивши `<documentation>` факультативні елементи:

```
<description targetNamespace=" xs:anyURI " >

    .  .  .
   <interface name=" xs:NCName "
           extends=" list of xs:QName "?
           styleDefault=" list of xs:anyURI "?  >

     <fault name=" xs:NCName "
             element=" xs:QName "?  >
     </fault>*
```

```
    <operation name=" xs:NCName "
           pattern=" xs:anyURI "
           style=" list of xs:anyURI "?
           wsdlx:safe=" xs:boolean "?  >

      <input messageLabel=" xs:NCName "?
           element=" union of xs:QName, xs:Token "?  >
      </input>*

      <output messageLabel=" xs:NCName "?
           element=" union of xs:QName, xs:Token "?  >
      </output>*

      <infault ref=" xs:QName " messageLabel=" xs:NCName "? > </infault>*

      <outfault ref=" xs:QName " messageLabel=" xs:NCName "? > </outfault>*

    </operation>*

  </interface>*
   .  .  .

</description>
```

Елемент `інтерфейсу` має два необов'язкових атрибути: `styleDefault` і `розширюється`. `StyleDefault` атрибут може бути використаний для визначення значення за замовчуванням для `стилю атрибутів` всіх операцій під цей інтерфейс (див. WSDL 2.0 Частина 1 "styleDefault пункту інформацію атрибут"). Атрибут `поширюється` для наслідування, і пояснюється наступним.

2.4.2 Успадкування інтерфейсу

Факультативний `поширюється` атрибутом інтерфейсу дозволяє розширювати чи наслідувати від одного або декількох інших інтерфейсів. У таких випадках інтерфейс містить операції інтерфейсів і поширюється разом з будь-якими операціями. Дві речі про розширення інтерфейсів заслуговують певної уваги.
По-перше, спадкування циклу (або нескінченної рекурсії) забороняється: інтерфейси, що розширюють дані інтерфейс розширюються самі по собі і не повинні продовжити цей інтерфейс або прямо, або побічно.
По-друге, ми повинні пояснити, що відбувається, коли операції з двох різних інтерфейсів мають ті ж імена цілі та ім'я операції. Є два випадки: або компоненти моделі операцій такі ж, або вони різні. Якщо компонент моделей одні й ті ж (за алгоритм порівняння компонента визначено в WSDL 2.0 частина 1 [*WSDL 2.0 Core*] "*Еквівалентність компоненти*"), то вони вважають ту ж операцію, тобто вони перетворюються в одну операцію, а також той факт, що вони були включені кілька разів не вважається помилкою. (Для операцій, компонент еквівалентності по суті означає, що дві операції мають однаковий набір атрибутів і нащадків.) У другому випадку, якщо дві операції з однаковим ім'ям в тому ж WSDL 2.0 цільовий простір імен, але не є еквівалентними, то це помилка. З наведених вище причин, це вважається хорошою практикою, щоб всі операції в рамках одного цільового імені названі однозначно.
Нарешті, тому що несправність також може бути визначена як елемент інтерфейсу (як описано в наступних розділах), те ж ім'я зіткнення правила застосовуються до цих конструкцій.
Скажімо, готель Greath хоче підтримувати стандартні операції журналу повідомлень для всіх вхідних повідомлень. Він хоче, щоб ця операція була багаторазовою по всій

системі бронювання, так що кожна служба буде посилати для потенційного використання службі обліку, зміст кожного повідомлення, він отримує разом з міткою часу і автором повідомлення. Одним із способів задоволення таких вимог є визначення журналу операцій в інтерфейсі, які можуть бути успадковані іншими інтерфейсами. Припускаючи messageLog елементів вже визначені в ghns імен з необхідним змістом, уразі спадкування використання показано в наступному прикладі. У результаті успадкування, `reservationInterface` тепер містить дві операції: `opCheckAvailability` і `opLogMessage`

Example 2-11. Interface Inheritance

```
 <description ...>
          ...
         <interface  name = "messageLogInterface" >

                 <operation name="opLogMessage"
                                 pattern="http://www.w3.org/ns/wsdl/out-only">
                         <output messageLabel="out"
                                 element="ghns:messageLog" />
                 </operation>

         </interface>

         <interface  name="reservationInterface"

extends ="tns:messageLogInterface" >

                 <operation name="opCheckAvailability"
                                 pattern="http://www.w3.org/ns/wsdl/in-out"
                                 style="http://www.w3.org/ns/wsdl/style/iri"
                                 wsdlx:safe = "true">
                        <input messageLabel="In"
                                 element="ghns:checkAvailability" />
                        <output messageLabel="Out"
                                 element="ghns:checkAvailabilityResponse" />
                        <outfault ref="tns:invalidDataFault"
messageLabel="Out"/>

                 </operation>
         </interface>
          ...
 </description>
```

Тепер давайте подивимося на елемент `інтерфейсу`, починаючи з `несправності`.

2.4.3 Інтерфейс несправності

`Несправність` елемента використовується, щоб оголосити несправності, які можуть виникнути в ході виконання операцій по інтерфейсу. Вони оголошуються безпосередньо під `інтерфейс`, і посилання з операцій, в яких вони застосовуються, з тим щоб забезпечити повторне використання в декількох операціях.
Розломи дуже схожі на повідомлення, і можуть розглядатися як особливий вид повідомлення. Обидві помилки та повідомлення можуть нести корисне навантаження, яке зазвичай описується елементом декларації. Однак, WSDL 2.0 лікує і повідомляє про несправності трохи інакше. Повідомлення операції прямого посилання на їх елементів з декларацією, проте недоліки операції опосередковано стосуються їх елементів декларації через несправності елемента, який визначений в інтерфейсі.
Для виявлення причини несправності на рівні інтерфейсу є надання їх повторного використання у кількох операціях. Ця конструкція особливо корисна при визначенні прив'язки, так як в обов'язкових розширень є додаткова інформація, яка пов'язана з

розломами. У разі SOAP, розломи мають коди і додаток до корисного навантаження. Шляхом визначення несправності на рівні інтерфейсу, загальних кодів і субкод можуть бути пов'язані з ними, забезпечуючи тим самим послідовність всіх операцій, які використовують його недоліки
`Несправність` елеменат має обов'язковий атрибут, `ім'я` якого має бути унікальним в межах елементу `інтерфейсу`, і дозволяє йому посилатися з експлуатації декларацій. Необов'язковий атрибут `елемент` може бути використаний для позначення схеми для утримання або корисного навантаження повідомлення про помилку. Її значення повинне бути QName глобальних елементів, визначених у розділі `типів`. Врахуйте, що при інших системах типу використовуються додаткові атрибути, можливо, необхідно буде визначити через WSDL 2.0 "з атрибутом розширення механізму, що дозволяє схему бути пов'язано з несправністю.

2.4.4 Інтерфейс операції

Як було показано раніше, `операція` елемент використовується для позначення операції і підтримуються інтерфейсом. Він асоціюється зі схемами повідомлення шаблон обміну повідомленнями (МЕФ), для абстрактного опису простої взаємодії з веб-службами.

2.4.4.1 Атрибути операції

`Операція` має два обов'язкових атрибути і один необов'язковий атрибут:

• обов'язкове `ім'я` атрибуту, як видно вже, повинно бути унікальним в межах інтерфейсу.

• необхідна `модель` атрибуту, значення якого повинно бути абсолютним URI, який ідентифікує бажаний МООСа для цієї `операції`. Європарламентарії додатково роз'яснюється в розумінні **2.4.4.3 Message Exchange Patterns (Європарламентарії).**

• Факультативний `стильовий` атрибут, значенням якого є список абсолютних URI. Кожен URI ідентифікує певний набір правил, які використовувалися при визначенні цієї `операції`. Це помилка, якщо не вказано певного стилю, але пов'язані з нею правила не дотримуються. [*WSDL 2.0 придатків*] визначає набір стилів.

○ Про Стиль RPC. RPC стиль обраний, коли `стиль` присвоюється значення http://www.w3.org/ns/wsdl/rpc. Він накладає обмеження для Remote Procedure Call-типи взаємодій.

○Про Стиль IPI. IPI стиль обраний, коли `стиль` присвоюється значенню http://www.w3.org/ns/wsdl/style/iri. Він накладає обмеження на повідомлення визначень, щоб вони могли бути серіалізовані щось на зразок HTTP URL закодовані.

○Про Multipart стиль. Multipart стиль обраний, коли `стиль` присвоюється значенню http://www.w3.org/ns/wsdl/style/multipart. У HTTP обов'язковими, для клієнтів XForms, повідомлення має бути визначений після Multipart .Ви можете знайти більш детальну інформацію про ці WSDL 2.0 Попереднє визначення стилі. Розділ **4.3 RPC Стиль** являє собою приклад використання `стилю` RPC. [*WSDL добавки 2.0*] наводяться приклади для стилю IPI і багатошарового стилю.
Зверніть увагу, що [*WSDL добавки 2,0*] надає зумовлені розширення для індикації безпеки експлуатації. `Wsdlx: безпечний` глобальний атрибут, значення якого може бути використане з операціями, чи буде операція стверджував "безпеку" (як визначено в розділі 3.5 Веб-архітектура [*Веб-архітектура*]) для клієнтів, для запуску. По суті, безпечною експлуатацією є будь-яка операція, яка не дає клієнтові ніяких нових зобов'язань. Наприклад, операція, яка дозволяє клієнту, щоб перевірити ціни на продукти як правило, не буде зобов'язувати клієнта купувати ці продукти, і, отже, повинна бути безпечною, у той час як операція на придбання продуктів буде зобов'язувати клієнтів платити за продукти які були замовлені, що, і, отже, не буде в безпеці.

Операція повинна бути, позначена як безпечна (за допомогою `wsdlx: безпечні` і, встановивши його значення "True"), якщо вона відповідає критеріям для безпечної взаємодії, визначених у розділі 3.5 Веб-архітектура [*Веб-архітектура*], оскільки це дозволяє інфраструктурі виконувати оптимізацію ефективності, такі, як попередньої вибірки, знову вибірки і кешування.
За замовчуванням значення цього атрибуту є помилковим. Якщо це помилково або не встановлено, то не зробив твердження про безпеку операції, операція може бути не безпечною.

2.4.4.2 Операція повідомлення списку

`Операція` буде також `введення, висновок, infault`, та / або `outfault` елементів, які визначають і звичайні типи несправності повідомлення, яке буде використовуватися за цією операцією. MEP передбачений `шаблон` атрибутом визначає, який з цих елементів повиннен бути включений, тому що кожен має MEP наповнювачі для типів повідомлень.
Ці операції вже розглядалися в **2.1.4 Визначення інтерфейсів**, у цьому розділі будуть лише прокоментовані додаткові можливості, які раніше були не пояснені.

2.4.4.2.1 MESSAGELABEL АТРИБУТИ

MessageLabel атрибутів вхідних та вихідних елементів не є обов'язковим. Це не потрібно явно встановити messageLabel при MOOCa у використанні є одним з восьми Європарламентарії визначених у WSDL 2.0 частина 2 [WSDL добавки 2,0] і має тільки одне повідомлення в заданому напрямку.

2.4.4.2.2 ЕЛЕМЕНТ АТРИБУТИ

Атрибут `елемента вхідних` та `вихідних` елементів використовується для визначення схеми контролю вмісту (AKA Деструктивна схема), коли зміст моделі визначається за допомогою XML Schema. Як ми вже бачили, воно може вказати QName елемента схеми, яка була визначена в розділі `типів`. Однак, як альтернативу він може вказати один з таких:

- **`Будь-який`**
  Зміст повідомлення будь-якого окремого елемента.
- **`None`**
  Немає змісту повідомлення, тобто повідомлення Деструктивно порожнє.
- **`Інші`**
  Зміст повідомлення описує Non-XML тип системи. Розширення атрибутів вказує тип.

`Елемент` атрибут також є факультативним. Якщо він не вказаний, то зміст повідомлення описується Non-XML тип системою.
Зверніть увагу, що існують ситуації, які передавши інформацію в `елементі` ознаки не є достатнім для реалізації сервісу, щоб однозначно визначити вхідні повідомлення і направити її у відповідні операції. У таких ситуаціях, додаткові кошти можуть бути необхідні для виявлення допомоги вхідного повідомлення. Див **5.1 дозволяючи легко Відправлення повідомлення** більш докладно.

2.4.4.2.3 MULTIPLE INFAULT АБО OUTFAULT ЕЛЕМЕНТИ

Коли `infault` та / або `outfault` зустрічаються кілька разів протягом операції, вони визначають альтернативні повідомленнях несправності.

2.4.4.3 розумінні Message Exchange Patterns (Європарламентарії)

WSDL 2.0 обміну повідомленнями шаблонів (Європарламентарії) використовуються для визначення послідовності і потужності анотації повідомлень в експлуатацію. По своїй конструкції, WSDL 2.0 Європарламентарії є абстрактними. Перш за все, вони конкретизують типи повідомлень. Європарламентарії визначили наповнювачів для повідомлень, і заповнювачів, пов'язаних з конкретними типами повідомлень, коли визначена операція, яка включає в себе вказуючи, які MEP використовувати для цієї операції. По-друге, якщо явно не вказано інше, наприклад, часу між повідомленнями, будь то картина синхронним або асинхронним, і чи є повідомлення відправленними за допомогою одного або декількох каналів.
Варто відзначити, що WSDL 2.0 Європарламентарії НЕ вичерпно описує безліч повідомлень, які можуть бути обмінені між послугами та іншими вузлами. За деякими попередніми угодами, інший вузол і / або послуг може відправити інші повідомлення (один з одним або з іншими вузлами), які не описані в MOOCa. Наприклад, хоча MEP може визначити одне повідомлення, відправлене від служби в жодний інший вузол, послуги визначається, що МВП може багатоадресно, що повідомлення на інші вузли. Для максимального повторного використання, WSDL 2.0 шаблонів обміну повідомленнями, визначає мінімальний контракт між сторонами та іншими веб-службами, і містить тільки інформацію, що має відношення як до веб-служби і клієнта, який займається цією службою.
У цілому вісім Європарламентарії визначені в [*WSDL добавки 2.0*]. Ці депутати Європарламенту повинні охоплювати найбільш поширені випадки використання, але вони не претендують на вичерпний список Європарламентарії. Детальніше Європарламентарії можуть бути визначені для конкретного застосування потреб зацікавлених сторін. (Див. **2.4.4.3 розумінні Message Exchange Patterns (Європарламентарії)**)
За вісім Європарламентарії визначається WSDL 2.0, деякі з них є варіаціями іншим, на підставі яких недоліків можуть бути отримані. Наприклад, в "Тільки шаблон (" http://www.w3.org/ns/wsdl/in-only ") складається рівно з одного повідомлення, отримані послуги від деяких інших вузлів. Як приклад In-Тільки, надійними в тільки шаблон ("http://www.w3.org/ns/wsdl/robust-in-only") також складається рівно з одного повідомлення, отриманої послуги, однак у цьому випадку несправності може бути викликане повідомлення і повинні бути доставлені за упорядником повідомлення. Якщо немає шляху до цього вузла, то вина повинна бути відкинута. Подробиці про загальну вину моделі, що використовуються на вісім WSDL 2.0 депутати Європарламенту, см. [*WSDL добавки 2.0*].
Залежно від того, як приступив до першого повідомлення в MOOCa, вісім WSDL 2.0 Європарламентарії можуть бути розділені на дві групи: в рамками Європарламенту, для яких служба отримує перше повідомлення в обмін та вихідного Європарламенту, за які служба посилає перше повідомлення в обміні. (Таке групування не передбачене в WSDL 2.0 і представлені тут тільки з метою зручності).
Часті питання про вихідні Європарламентарії служба знає, де можна відправити повідомлення. Служби, які використовують вихідні Європарламентарії звичайно є частиною великомасштабної інтеграції систем, які спираються на відображення об'єктів і маршрутизації. У таких системах, що виходить Європарламентарії корисні для визначення функціональних служб абстрактно, в тому числі його вимоги до потенційних клієнтів, а інформація, адреса кінцевої точки може бути надана в розгортанні або час виконання основних інтеграційних інфраструктур. Наприклад, застереження Greath готель система може вимагати, щоб кожен раз, коли клієнт взаємодіє з системою, щоб перевіряв наявність даних про клієнта повинні увійти по CRM системі. На етапі проектування, невідомо, яка зокрема система CRM буде використовуватися разом з системою бронювання. Для задоволення цієї вимоги, ми можемо змінити "reservationInterface" у Приклад 2-1 і включити вихідної logInquiry операції. Ця операція `logInquiry` рекламує потенційним клієнтам послугу клієнтові,

що дані будуть доступні під час виконання. При замовленні послуги розгорнуті в IT-ландшафт Greath's, відповідної конфігурації під час запуску і час інфраструктура допоможе визначити, яка система CRM буде отримувати дані про клієнтів та журналу їх відповідним чином. Варто відзначити, що на додаток до використовуваної CRM системи для клієнтів, ті самі дані можуть також використовуватися системою аналізу продуктивності інструменту для різних цілей. Надання вихідної операції на замовлення послуги дає можливість вільного з'єднання і так поліпшує загальний Greath IT-ландшафт гнучкість і масштабованість.
Example 2-12. Use of outbound MEPs

```
 <description ...>
          ...
         <interface  name="reservationInterface">
                   ...
                  <operation name="opCheckAvailability" ...  >

                  <operation name="opLogInquiry"
                                 pattern="http://www.w3.org/ns/wsdl/out-only"
>
                          < output messageLabel="Out"
element="ghns:customerData"  />
                  </operation>

         </interface>
          ...
 </description>
```

Хоча вісім Європарламентарії визначено в WSDL 2.0 Частина 2 [*WSDL 2.0 придатків*] призначені для покриття більшості випадків використання, WSDL 2.0 розроблений цей набір буде розширюватися. Ось чому Європарламент при визначені URI, має не фіксований набір ознак.
Додаткову інформацію про визначення нових даних Європарламенту,**. 4.2 Визначення нових даних.**

2.5 Детальніше про прив'язку

Bindings використовується для постачання протоколом та кодуванням подробиць, які визначають, як повідомлення повинні бути відправлені або отримані. Кожен `обов'язковий` елемент використовує певне *обов'язкове* розширення вказуючи таку інформацію. WSDL 2.0 Частина 2 [*WSDL 2.0 придатків*] визначає кілька обов'язкових розширень, які звичайно використовуються. Однак обов'язковими розширеннями, які не визначені в WSDL 2.0 Частини 2 також можуть бути використані за умови, що клієнт і обслуговування інструментальних засобів їх підтримує.
Binding інформація повинна бути представлена для кожної операції в інтерфейсі, який використовується в кінцевій точці. Однак, якщо необхідно обов'язкове розширення, то ця інформація буде необхідна тільки явні, представленої на рівні інтерфейсу, а також порушуючи правила буде поширюватися неявна інформація в операціях інтерфейс. Див, наприклад, за замовчуванням обов'язковими нормами SOAP обов'язковим розширенням WSDL 2.0 частина 2 [*WSDL добавки 2.0*].

2.5.1 Синтаксис резюме для прив'язок

Після прив'язки задаються за допомогою розширень для мови WSDL 2.0 (тобто обов'язкового продовження не в WSDL 2.0 імен), XML для вираження обов'язковим буде складатися із суміші елементів і атрибутів з WSDL 2.0 імен та імен з обов'язковим розширенням , з використанням WSDL 2.0 "з відкритої моделі змісту. Ось синтаксис резюме для обов'язкового, спрощеного, пропустивши необов'язкові елементи документації. Майте на увазі, що цей синтаксис резюме показує тільки

елементи і атрибути, визначені в просторі імен WSDL 2.0. Коли визначено фактично обов'язкові елементи і атрибути з простору імен бажаного обов'язкового розширення також будуть залучені як це передбачено в зокрема, що обов'язкове розширення.

```
<description targetNamespace=" xs:anyURI " >
    .  .  .
   < binding  name=" xs:NCName " interface=" xs:QName "?  >

     <fault ref=" xs:QName " >  </fault>*

     <operation ref=" xs:QName " >
       <input messageLabel=" xs:NCName "? > </input>*
       <output messageLabel=" xs:NCName "? > </output>*
       <infault ref=" xs:QName " messageLabel=" xs:NCName "? > </infault>*
       <outfault ref=" xs:QName " messageLabel=" xs:NCName "? > </outfault>*
     </operation>*

   </ binding >*
    .  .  .
 </description>
```

`Обов'язковий` синтаксис Parallels синтаксис `інтерфейсу`: кожен інтерфейс має обов'язкового партнера. Незважаючи на це, вони дійсно мають різні конструкції, оскільки вони знаходяться в різних просторах символів і призначені для різних цілей.

2.5.2 Багаторазові Bindings

Обов'язковий інтерфейс може бути багаторазовим (стосовно будь-який інтерфейс) або одноразового використання (вказуються для певного інтерфейсу). Одноразового використання прив'язки можуть бути зазначені у деталізації інтерфейсу (за умови обов'язкового розширення забезпечує відповідні правила), або на операцію за основу при необхідності. Одноразового використання обов'язково була продемонстрована в **2.1.5 Визначення Binding**.
Щоб визначити, `обов'язковий` елемент просто опускається інтерфейс атрибутів і опускає вказівки яких-небудь конкретних операцій і конкретно обов'язкових деталей. Кінцеві згодом можуть послатися на багаторазові обов'язки в тому ж порядку, як і для одноразового використання. Таким чином, стають обов'язковими багаторазові, пов'язані з певним інтерфейсом, коли на нього посилається від кінцевої точки, тому що кінцевою точкою є частина служби, та служба задає конкретний інтерфейс, який він реалізує. Після багаторазового використання обов'язкових не вказується інтерфейс, багаторазові прив'язки не можуть вказати конкретні операції докладніше. Таким чином, багаторазові прив'язки можна визначити тільки за допомогою обов'язкових розширень, які мають відповідні правила, такі, що обов'язкова інформація тільки має бути явним, представленою на рівні інтерфейсу.

2.5.3 Binding несправності

`Обов'язковий` Associates несправність конкретного формату повідомлення з абстрактним інтерфейсом. Він описує, як помилки, які відбуваються в рамках обміну повідомленнями операції будуть відформатовані, тому що провина не виникає сама по собі. Скоріше, вина відбувається, як частина обміну повідомленнями передбачених інтерфейсом операції і обов'язкові `роботи`.
Обов'язкова `провина` має один обов'язковий атрибут, який Ref це посилання, по QName, до `інтерфейсу вини`. Він визначає абстрактний інтерфейс `вини`, для яких обов'язкова інформація уточнюється. Майте на увазі, що значення атрибута `ref` всі несправності у відповідності з обов'язковими повинні бути унікальними. Тобто, ніхто не може визначити декілька прив'язок з тим же інтерфейсом вини протягом певного `обов'язку`.

2.5.4 Binding Operations

Обов'язкова `операція` описується конкретною прив'язкою інтерфейсу операції по конкретних форматах повідомлення. Інтерфейс операція однозначно ідентифікується WSDL 2.0 цільовим імен інтерфейсу і назва операції в рамках цього інтерфейсу, через обов'язковий атрибут `ref` обов'язкових `операцій`. Як і у випадку несправності, для кожної `операції` в `обов'язковому`, значення атрибуту `ref` повинно бути унікальним.

2.5.5 Розширення SOAP Binding

WSDL, SOAP 2.0 Binding продовження (див. WSDL 2.0 частина 2 [*WSDL добавки 2.0*]) була призначена головним чином для підтримки особливостей SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework*]. Однак, для забезпечення сумісності, але й забезпечує деяку підтримку SOAP 1.1 [*SOAP 1.1*].
Наприклад, за допомогою WSDL, SOAP 2.0 обов'язковим розширенням вже була представлена в **2.1.5 Визначення сили**, але деякі додаткові моменти заслуговують на увагу:

- Так само обов'язкове розширення використовується як для SOAP 1.2, SOAP 1.1, `wsoap: версія` атрибута при умові, щоб вказати, яку версію ви хочете SOAP. Якщо цей атрибут не вказаний, то за замовчуванням SOAP 1.2.
- WSDL, SOAP 2.0 обов'язковим розширенням визначає набір правил за замовчуванням, так що прив'язка може бути вказана на рівні інтерфейсу або на рівні експлуатації (або обидва), з урахуванням рівня експлуатації, який має пріоритет. Однак воно не визначає обов'язкові правила за умовчанням для помилки. Таким чином, якщо даний інтерфейс визначає будь-яку несправність, то відповідна обов'язкова інформація повинна бути чітко передбачена для кожної такої провини.
- Якщо HTTP використовується в якості основного протоколу, то обов'язковим є контроль кожної операції і буде використовувати HTTP GET або POST. (Див. **2.5.7 HTTP GET Versus POST: Який використовувати?**.)
  Ось приклад, який показує, як SOAP 1.2 обов'язковим (як показано вище) і SOAP 1.1 обов'язковими.

Example 2-13. SOAP 1.2 and SOAP 1.1 Bindings

```
<?xml version="1.0" encoding="utf-8" ?>
<description
  xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace="http://greath.example.com/2004/wsdl/resSvc"
  xmlns:tns="http://greath.example.com/2004/wsdl/resSvc"
  xmlns:ghns="http://greath.example.com/2004/schemas/resSvc"
  xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">

  ....

  <!-- SOAP 1.2 Binding -->
  <binding name="reservationSOAPBinding"
    interface="tns:reservationInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response"/>
```

```
    <fault ref="tns:invalidDataFault"
      wsoap:code="soap:Sender"/>

  </binding>

  <!-- SOAP 1.1 Binding -->
  <binding name="reservationSOAP11Binding"
    interface="tns:reservationInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:version="1.1"
    wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP/">

    <operation ref="tns:opCheckAvailability"/>

    <fault ref="tns:invalidDataFault"
      wsoap:code="soap11:Client"/>

  </binding>


  <service name="reservationService"
    interface="tns:reservationInterface">

    <!-- SOAP 1.2 End Point -->
    <endpoint name="reservationEndpoint"
      binding="tns:reservationSOAPBinding"
      address="http://greath.example.com/2004/reservation"/>

    <!-- SOAP 1.1 End Point -->
    <endpoint name="reservationEndpoint2"
      binding="tns:reservationSOAP11Binding"
      address="http://greath.example.com/2004/reservation"/>

  </service>
</description>
```

2.5.5.1 Пояснення прикладу

Більшість ліній в даному прикладі, так само, як вказувалося раніше в **2.1.5 Визначення прив'язки**, тому ми лише вкажемо на лінії, які демонструють щось нове для SOAP 1.1 обов'язковими.

`<Опис ... xmlns: soap11 = "http://schemas.xmlsoap.org/soap/envelope/">`

Цей простір імен для термінів, визначених у SOAP 1.1 Specification [*SOAP 1.1*].

`<Обов'язкового wsoap: Version = "1.1"`

Цей рядок показує, що це обов'язкове використання SOAP 1.1 [*WSDL 2.0 SOAP 1.1 обов'язкову*], а не SOAP 1.2.

`wsoap: протокол = "http://www.w3.org/2006/01/soap11/bindings/HTTP/">`

Цей рядок визначає, що HTTP повинні використовуватися в якості основного протоколу передачі. Див також **2.5.7 HTTP GET Versus POST: Яку використовувати?** .

`<operation ref="tns:opCheckAvailability"/>`

Зверніть увагу, що `wsoap: mep` це не відноситься до обов'язкових SOAP 1.1.

`<fault...wsoap:code="soap11:Client"/>`

Цей рядок визначає, SOAP 1.1 код несправності, який буде використаний при передачі invalidDataFault.

2.5.6 HTTP Binding Розширення

На додаток до WSDL, SOAP 2.0 обов'язковим розширенням описаної вище, WSDL 2.0 Частина 2 [WSDL 2.0 придатків] визначає обов'язкове розширення для HTTP 1.1 [*IETF RFC 2616*] і HTTPS [*IETF RFC 2818*], так що ці протоколи можуть бути

використані для відправлення й одержування повідомлення, перше кодування їх у SOAP. HTTP обов'язковим розширенням надає безліч можливостей для контролю:

- Яка HTTP операція буде використана. (GET, PUT, POST, DELETE та інші операції HTTP підтримується)
- введення, висновок і серіалізаціяпровини
- передача кодувань
- аутентифікація вимоги
- Печиво
- HTTP над TLS (HTTPS)

Як і в WSDL 2.0 SOAP обов'язковим розширенням HTTP також надає правила, що дозволяють обов'язковій інформації, яка буде вказана на рівні інтерфейсу і використовувати за замовчуванням для кожної операції в постраждалих інтерфейсах, проте.Ось приклад використання HTTP обов'язкового розширення, щоб перевірити наявність вільних номерів у готелі Greath.

Example 2-14. HTTP Binding Extension

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
       .  .  .
      xmlns:whttp="http://www.w3.org/ns/wsdl/http" >

   .  .  .
  <binding name="reservationHTTPBinding"
      interface="tns:reservationInterface"
      type="http://www.w3.org/ns/wsdl/http"
      whttp:methodDefault="GET">

    <operation ref="tns:opCheckAvailability"
      whttp:location="{checkInDate}"  />
  </binding>

  <service name="reservationService"
    interface="tns:reservationInterface">

    <!-- HTTP 1.1 GET End Point -->
    <endpoint name="reservationEndpoint"
      binding="tns:reservationHTTPBinding"
      address="http://greath.example.com/2004/checkAvailability/"/>

  </service>
   .  .  .
</description>
```

2.5.6.1 Пояснення прикладу

Більша частина цього прикладу така ж , як вказувалося раніше в **2.1.5 Визначення прив'язки**, тому ми лише вказуємо на те, яке демонструює щось нове для HTTP обов'язковим розширенням.

<description...xmlns:whttp="http://www.w3.org/ns/wsdl/http">

Це визначає префікс імен для елементів і атрибутів, визначених у WSDL 2.0 HTTP обов'язковим розширенням.

<Обов'язкового type = "http://www.w3.org/ns/wsdl/http"

Це оголошує обов'язковою для HTTP.

whttp: methodDefault = "GET">

Метод за замовчуванням для операцій в даному інтерфейсі буде HTTP GET.

whttp: location = "() checkInDate">

Whttp: розташування атрибуту задає шаблон для перетворення вхідних даних повідомлень інстанції в дорозі компоненті URI запиту. Обов'язкові правила за умовчанням для HTTP вказують, що серіалізація для введення за замовчуванням є застосування GET x-www-форм-urlencoded. Фігурні дужки використовуються, щоб вказати ім'я схеми типу в схемі вхідне повідомлення, яке визначає, що введення даних інстанції будуть вставлені в дорозі компоненті URI запиту. Фігурні дужки додаю ім'я замінене екземпляром даних при побудові компонента шляху , що залишився, (не передбачених whttp: location) будуть або серіалізованний до запиту частині рядка URI або в тілі повідомлення, а саме: якщо "/" додається до фігурної дужки-закритого типу ім'я, то всі залишилися дані повідомлення введення інстанції будуть серіалізовані в тексті листа. В іншому випадку це буде серіалізованний в параметри запиту.

Таким чином, у цьому прикладі, кожен з елементів у tCheckAvailability типу будуть серіалізовані в параметри запиту. Приклад результаті URI б бути http://greath.example.com/2004/checkAvailability/5-5-5?checkOutDate=6-6-5&roomType=foo.

Ось альтернативний приклад додає, що "/" на ім'я типу для серіалізації залишилися екземпляри даних в тілі повідомлення:
Приклади 2-15. Сериализация підмножина типів в дорозі

```
. . .
<operation ref="tns:opCheckAvailability"
    whttp:location="bycheckInDate/{checkInDate/}"  >
. . .
```

Це буде замість серіалізації в URI запиту, такі як: http://greath.example.com/2004/checkAvailability/bycheckInDate/5-5-5. Інша частина вмісту повідомлень піде в тіло повідомлення HTTP.

2.5.7 HTTP GET Versus POST: Яку використовувати?

При обов'язковому використанні HTTP вказуючи на операцію, WSDL 2.0 автор повинен вирішити, який HTTP метод доцільно використовувати - як правило, вибір між GET та POST. У контексті Інтернету в цілому (а не конкретно веб-послуг), W3C Технічна архітектура Group (TAG) розглядає питання про те, коли це доречно використовувати GET, в порівнянні коли використовувати пост в пошуку URI, звернення, *а також використання HTTP GET та* POST ([*W3C TAG Висновок: Використання HTTP GET*]).Аанотація:
"... Проектувальники повинні приймати [Get] для безпечних операцій, таких як прості запити. POST підходить для інших типів додатків, де користувач прохання має потенціал для зміни стану ресурсу (або пов'язані з ними ресурси). Знаходження пояснює як зробити вибір між HTTP GET та POST для застосування з урахуванням архітектурних, безпечних та практичних міркувань ".
Нагадаємо, що концепцію безпечної експлуатації було обговорено в **2.4.4.1 Атрибути операції**. (Резюме, безпечну операцію, яка не викликає нові зобов'язання.) Хоча wsdlx: безпечний атрибут інтерфейсу операція показує, що абстрактна операція знаходиться в безпеці, він автоматично не використаний в HTTP рівні, коли зазначений обов'язковий характер. Вибір отримати або посади визначається на обов'язковий рівень:

- Якщо WSDL, SOAP 2.0 обов'язкового розширення використовується (**2.5.5 Розширення SOAP Binding**), з HTTP як основний транспортний протокол, а потім може бути вказано значення:
  wsoap: протокол = "http://www.w3.org/2003/05/soap/bindings/HTTP/"
  з обов'язковим елементом (вказати використання HTTP як основний протокол), а також
  wsoap: MEP = "http://www.w3.org/2003/05/soap/mep/soap-response/"
  з обов'язковим елементом операції, яка викликає GET, що буде використовуватися за замовчуванням.
- Якщо WSDL 2.0 HTTP обов'язкового розширення використовується безпосередньо (**2.5.6 HTTP Binding Extension**), GET можуть бути визначені шляхом встановлення або:
  whttp: methodDefault = "GET"
  з обов'язкового елементу, або
  whttp: метод = "GET"
  з обов'язковим елементом операції, яка перекриває whttp: methodDefault якщо встановити на обов'язковий елемент; або
  wsdlx: безпечні = "true"
  на кордоні операції інтерфейсу. Коли два вищезгадані пункти прямо не встановлено, і коли операція пов'язана інтерфейс, позначена як безпечні, HTTP Binding буде за умовчанням встановлено методом GET.
  Наприклад, у визначенні Greath інтерфейсі показано в прикладі 2-4, wsdlx: безпечні атрибуту встановлено значення "True". HTTP обов'язковим зазначенням на прикладі 2-14 можуть скористатися цим і бути спрощені, як показано нижче, і до цих пір методу HTTP GET значення за замовчуванням:
  Приклади 2-16. Безпека і HTTP Binding

```
<?xml version="1.0" encoding="utf-8" ?>

<binding name="reservationHTTPBinding"

      interface="tns:reservationInterface"

      type="http://www.w3.org/ns/wsdl/http" >

    <operation ref="tns:opCheckAvailability"

        whttp:location="{checkInDate}"/>

  </binding>
```

3. Advanced Topics I: Імпорт механізми

3.1 Importing WSDL

У деяких випадках автори WSDL можуть розділити опис веб-служби на два або декілька документів. Наприклад, при описі розробляє кілька авторів, то це зручно розділити її на кілька частин. Іншим дуже важливим є випадок, коли ви очікуєте частини опису для повторного використання в декількох контекстах. Ясно, що це небажано, щоб вирізати і вставити розділи одного документа в інший, оскільки це значною кількістю помилок і веде до підтримання проблем. Більш того, вам може знадобитися повторне використання компонентів, які відносяться до WSDL: TargetNamespace, який відрізняється від положення документа, який ви пишете, і в цьому випадку правила WSDL 2.0 перешкодять вам просто вирізати і вставити їх в документ.
Щоб вирішити ці проблеми, WSDL 2.0 надає два механізми для веб-документів

модулярізації Опис послуги: `імпорт` і `включення`. У цьому розділі обговорюються механізм імпорту та описані деякі типові випадки, коли воно може бути використано. `Імпортний` механізм дозволяє одним послатися на визначення веб-служби компонентів, які відносяться до інших просторів імен. Щоб проілюструвати це, розглянемо застереження Greath готельних послуг. Припустимо, що застереження, служба використовує стандартні кредитні картки перевірку служби, яка надає фінансові послуги. Крім того, припустимо, що компанії в сфері фінансових послуг вирішили, що вони будуть корисні для повідомлення про помилки у кредитних карт перевірка з використанням єдиного набору недоліків, і визначені ці недоліки в наступних описах веб-служби:

Example 3-1. Standard Credit Card Validation Faults (credit-card-faults.wsdl)

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"
         targetNamespace="http://finance.example.com/CreditCards/wsdl"
         xmlns:tns="http://finance.example.com/CreditCards/wsdl"
         xmlns:cc="http://finance.example.com/CreditCards/xsd">

   <documentation>
         This document describes standard faults for use
       by Web services that process credit cards.
   </documentation>

   <types>
         <xs:import xmlns:xs="http://www.w3.org/2001/XMLSchema"
             namespace="http://finance.example.com/CreditCardFaults/xsd"
             schemaLocation="credit-card-faults.xsd" />
   </types>

   <interface name="creditCardFaults">

         <fault name="cancelledCreditCard" element="cc:CancelledCreditCard">
             <documentation>Thrown when the credit card has been
cancelled.</documentation>
         </fault>

         <fault name="expiredCreditCard" element="cc:ExpiredCreditCard">
             <documentation>Thrown when the credit card has
expired.</documentation>
         </fault>

         <fault name="invalidCreditCardNumber"
element="cc:InvalidCreditCardNumber">
             <documentation>Thrown when the credit card number is invalid.
                 This fault will occur if the wrong credit card type is
specified.
           </documentation>
         </fault>

         <fault name="invalidExpirationDate"
element="cc:InvalidExpirationDate">
             <documentation>Thrown when the expiration date is
invalid.</documentation>
         </fault>

   </interface>

 </description>
```

Цей приклад визначає інтерфейс, `creditCardFaults`, який містить чотири помилки, `cancelledCreditCard, expiredCreditCard, invalidCreditCardNumber` і `invalidExpirationDate`. Ці компоненти належать до простору імен `http://finance.example.com/CreditCards/wsdl`.

Оскільки ці недоліки визначаються в різні WSDL: TargetNamespace, ніж той, що

використовується службою Опис Greath веб-імпорту повинен бути використаний, щоб зробити їх доступними в рамках послуги Опис Greath Інтернеті, як показано в наступному прикладі:

Example 3-2. Using the Standard Credit Card Validation Faults (use-credit-card-faults.wsdl)

```
  <? XML Version = "1.0"?
 <description
        targetNamespace="http://greath.example.com/2004/wsdl/resSvc"
        xmlns:ghns="http://greath.example.com/2004/schemas/resSvc"
        xmlns:cc="http://finance.example.com/CreditCards/wsdl"
        xmlns="http://www.w3.org/ns/wsdl"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <documentation>
                Description: The definition of the reservation Web service of
                GreatH hotel. Author: Joe Somebody Date: 05/17/2004
        </documentation>

        <import namespace="http://finance.example.com/CreditCards/wsdl"
              location="credit-card-faults.wsdl"/>
         .  .  .
        <interface name="reservation" extends="cc:creditCardFaults">
                 .  .  .
                <operation name="makeReservation"
                        pattern="http://www.w3.org/ns/wsdl/in-out">

                        <input messageLabel="In"
element="ghns:makeReservation" />

                        <output messageLabel="Out"
                                element="ghns:makeReservationResponse" />

                        <outfault ref="invalidDataFault" messageLabel="Out"
/>

                        <outfault ref="cc:cancelledCreditCard"
messageLabel="Out" />
                        <outfault ref="cc:expiredCreditCard"
messageLabel="Out" />
                        <outfault ref="cc:invalidCreditCardNumber"
messageLabel="Out" />
                        <outfault ref="cc:invalidExpirationDate"
messageLabel="Out" />
                </operation>
        </interface>
 </description>
```

Служба бронювання готелів заявляє, що вона, використовуючи компоненти, з іншого простору імен за допомогою імпорту> елемент. Імпорт елементів має `обов'язковий` атрибут, який визначає імена інших імен, а `необов'язковий` атрибут дає процесору натяк, де можна знайти опис інших імен. `Бронювання` інтерфейс розширює `creditCardFault` інтерфейс від інших імен, з тим щоб наявні недоліки в застереження інтерфейсу. Нарешті, `makeReservation` операція відноситься до стандартної помилки у своїх `outfault` елементів.

Ще одна типова ситуація для використання імпорту є визначення стандартного інтерфейсу. Наприклад, припустимо, в готельному господарстві вирішили, що було б корисно мати стандартний інтерфейс для внесення застережень. Цей інтерфейс буде належати деяким іменам Асоціації промисловості, наприклад `http://hotels.example.com/reservations/wsdl`. Кожен готель, який здійснює стандартні послуги бронювання б визначили службу у свій власний простір імен, наприклад `http://greath.example.com/2004/wsdl/resSvc`. Опис кожного сервісу буде

імпортувати `http://hotels.example.com/reservations/wsdl` імен і відносяться до стандартного інтерфейсу застереження в ньому.

3.2 Importing Schemas

WSDL 2.0 документ може містити одну або більше схем XML визначено в `wsld: типи` елементів. Цей розділ показує правильний шлях для позначення цих схем, як в рамках одного документа та інших документів.

3.2.1 Схеми в документах імпортні

У цьому прикладі ми розглянемо деякі Greath Hotel веб-служби, пошуку та оновлення даного бронювання. Служба веб-пошуку визначається в `retrieveDetails.wsdl` WSDL 2.0 документ, поряд з схемою для формату повідомлення. Оновлення веб-служби визначено в `updateDetails.wsdl` WSDL 2.0 документ, який імпортує перший документ, і посилається на обидва WSDL 2.0 та схеми визначення, що містяться в імпортованому документі.
Приклад 3-3 показує визначення послуги веб-пошуку в просторі імен `http://greath.example.com/2004/services/retrieveDetails`. Це WSDL 2.0 Документ також містить вбудовану схему, яка описує замовлення в просторі імен `http://greath.example.com/2004/schemas/reservationDetails`. Ця схема видно `retrieveDetailsInterface` визначення інтерфейсу, який відноситься до нього до статті вихідного повідомлення операції.

Example 3-3. The Retrieve Reservation Details Web Service: retrieveDetails.wsdl

```
                                                    <?xml version="1.0"
encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/retrieveDetails"
         xmlns:tns="http://greath.example.com/2004/services/retrieveDetails"

xmlns:wdetails="http://greath.example.com/2004/schemas/reservationDetails"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">

         <documentation>
                 This document describes the GreatH Retrieve Reservation
Details
                 Web service.
         </documentation>

         <types>
                 <xs:schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://greath.example.com/2004/schemas/reservationDetails">

                         <xs:element name="reservationDetails">
                                 <xs:complexType>
                                         <xs:sequence>
                                                 <xs:element
name="confirmationNumber"
                                                         type="string" />
                                                 <xs:element
name="checkInDate" type="date" />
                                                 <xs:element
name="checkOutDate" type="date" />
                                                 <xs:element name="roomType"
type="string" />
                                                 <xs:element name="smoking"
type="boolean" />
                                         </xs:sequence>
```

```
                                </xs:complexType>
                        </xs:element>
                </xs:schema>
        </types>

        <interface name="retrieveDetailsInterface">

                <operation name="retrieve"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="#none" />
                        <output messageLabel="Out"
                                element="wdetails:reservationDetails" />
                </operation>

        </interface>

 </description>
```

Приклад 3-4 показує визначення з оновлення веб-служб в просторі імен `http://greath.example.com/2004/services/updateDetails`. `UpdateDetailsInterface` інтерфейс розширює `retrieveDetailsInterface`. Тим не менше, `retrieveDetailsInterface` належить до простору імен `http://greath.example.com/2004/services/retrieveDetails`, так `updateDetails.wsdl` необхідно імпортувати `retrieveDetails.wsdl` зробити цей простір імен видним. `UpdateDetailsInterface` інтерфейс також використовує `reservationDetails` елемент визначення, який міститься в вбудованій схемі імпортного `retrieveDetails.wsdl` документу. Однак ця схема не буде автоматично відображатися в `updateDetails.wsdl`. Щоб зробити його видимим, updateDetails.wsdl документ необхідно імпортувати простір імен у вбудовану схему в рамках типів елементів з використанням XML-схеми `імпорту` елементів.

У цьому прикладі, `schemaLocation` атрибут елемента `імпорту` був опущений. `SchemaLocation` атрибут є натяком на WSDL 2.0 процесора, який повідомляє йому, де шукати імпортованих імен схеми. Тим не менше, WSDL 2.0 процесором вже оброблено `retrieveDetails.wsdl` документ, який містить імпортовані імена у вбудованій схемі так що не потрібно ніяких натяків. Однак, ця поведінка залежить від реалізації процесорів і тому не може покластися.

Хоча документ WSDL 2.0 не має права опускати `schemaLocation` атрибут, краще забезпечити надійне значення для його або перемістити вбудовану схему у вигляді окремого документа, скажімо `reservationDetails.xsd`, так і безпосередньо імпортувати його в `типах` елементів обох `retrieveDetails.wsdl` і `updateDetails.wsdl`. Загалом, схем, які, як очікується, будуть посилатися з більш ніж одного WSDL 2.0 документа повинні бути визначені в окремому документі схемою, а не бути вбудовані.

Example 3-4. The Update Reservation Details Web Service: updateDetails.wsdl

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/updateDetails"
         xmlns:tns="http://greath.example.com/2004/services/updateetails"

xmlns:retrieve="http://greath.example.com/2004/services/retrieveDetails"

xmlns:details="http://greath.example.com/2004/schemas/reservationDetails"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">

         <documentation>
                 This document describes the GreatH Update Reservation Details
                 Web service.
         </documentation>

         <import
```

```
namespace="http://greath.example.com/2004/services/retrieveDetails"
                location="retrieveDetails.wsdl" />

        <types>
                <xs:import

namespace="http://greath.example.com/2004/schemas/reservationDetails" />
        </types>

        <interface name="updateDetailsInterface"
                extends="retrieve:retrieveDetailsInterface">

                <operation name="update"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In"
                                element="details:reservationDetails" />
                        <output messageLabel="Out"
                                element="details:reservationDetails" />
                </operation>

        </interface>

 </description>
```

3.2.2 Кілька вбудованих схем в одному документі

WSDL 2.0 документ може визначити кілька вбудованих схем у своїх `типах` елементів. Дві або більше схемами можуть мати однакові імена цільових умов, що вони не визначають ті ж елементи, або типи. Це помилка, щоб визначити той же тип елементу або більше разів, навіть якщо ці визначення ідентичні.
Кожний простір імен вбудовану схему стає видимим для визначення веб-служби. Тим не менше, імена автоматично не видно іншими схемами. Кожну вбудовану схему необхідно явно імпортувати в будь-які інші імена. `SchemaLocation` ознаки не потрібно в даному випадку, так WSDL 2.0 процесор знає місцезнаходження кожної схеми в силу того, оброблено 2.0 вміщають WSDL документа.
Щоб проілюструвати це, розглянемо Приклад 3-5 який містить дві вбудовані схеми. `http: / / greath.example.com/2004/schemas/reservationItems` імен містить деякі елементи, по пунктам, які з'являються в бронюваннях. `Http: / / greath.example.com/2004/schemas/reservationDetails` імен містить `reservationDetails` елемент, який посилається на пункт елементів. Схема для імен `http://greath.example.com/2004/schemas/reservationDetails` містить імпорт елементів, який імпортує `http://greath.example.com/2004/schemas/reservationItems` імена. `SchemaLocation` атрибути, необхідні для імпорту з цією схемою .

Example 3-5. Multiple Inline Schemas: retrieveItems.wsdl

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/retrieveDetails"
        xmlns:tns="http://greath.example.com/2004/services/retrieveDetails"

xmlns:wdetails="http://greath.example.com/2004/schemas/reservationDetails"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <documentation>
                This document describes the GreatH Retrieve Reservation
Details
                Web service.
        </documentation>

        <types>
```

```
                <xs:schema
targetNamespace="http://greath.example.com/2004/schemas/reservationItems">

                        <xs:element name="confirmationNumber" type="string"
/>
                        <xs:element name="checkInDate" type="date" />
                        <xs:element name="checkOutDate" type="date" />
                        <xs:element name="roomType" type="string" />
                        <xs:element name="smoking" type="boolean" />

                </xs:schema>

                <xs:schema
targetNamespace="http://greath.example.com/2004/schemas/reservationDetails"

xmlns:items="http://greath.example.com/2004/schemas/reservationItems">

                        <xs:import

namespace="http://greath.example.com/2004/schemas/reservationItems" />

                        <xs:element name="reservationDetails">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element
ref="items:confirmationNumber" />
                                                <xs:element
ref="items:checkInDate" />
                                                <xs:element
ref="items:checkOutDate" />
                                                <xs:element
ref="items:roomType" />
                                                <xs:element
ref="items:smoking" />
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:schema>

        </types>

        <interface name="retrieveDetailsInterface">

                <operation name="retrieve"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="#none" />
                        <output messageLabel="Out"
                                element="wdetails:reservationDetails" />
                </operation>

        </interface>

 </description>
```

3.2.3 The schemaLocation Атрибут

У наведених вище прикладах, схеми були визначені вбудовані в WSDL 2.0 документи. У цьому розділі обговорюється правильний спосіб визначити `schemaLocation` атрибути елемента імпорту схеми які забезпечує процесор з натяком для розміщення цих схем.
Приклад 3-4 показує, як один документ WSDL 2.0 імпорт схема визначена в іншій, тобто Приклад 3-3. Точно так само Приклад 3-5 показує, як одна схема, в документі WSDL 2.0 імпорт іншою схемою визначені у тому ж документі. В обох цих прикладах,

`schemaLocation` атрибутом був опущений з процесором 2,0 WSDL вважалося б знати, як знайти імпортовані схеми, оскільки вони є частиною WSDL 2.0 документа. `SchemaLocation` атрибут може бути використаний для процесорів посиланням URI, явно знаходить схемами. Посилання URI є URI плюс факультативний ідентифікатором фрагмента, який вказує частину ресурсу. Для схем, фрагментів потрібно визначити схему елементів. Найпростішим способом зробити це є використання ID атрибуту, однак XPointer (див. [*XPointer рамок*]) також можуть бути використані.

3.2.3.1 Використання ID атрибута для визначення вбудованих схем

У 3-6 прикладі показано використання атрибуту `ідентифікатора`. Вони мають вбудовані схеми `id` атрибути. Ідентифікатор `http://greath.example.com/2004/schemas/reservationItems` схеми пунктів і ID схеми `http://greath.example.com/2004/schemas/reservationDetails` це деталі. `Імпорт елементів` у схемі `http://greath.example.com/2004/schemas/reservationDetails` використовує ідентифікатор `http://greath.example.com/2004/schemas/reservationItems` схему в `schemaLocation` атрибутів, тобто кількість елементів .

Example 3-6. Using Ids in Inline Schemas: schemaIds.wsdl

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/retrieveDetails"
         xmlns:tns="http://greath.example.com/2004/services/retrieveDetails"

xmlns:wdetails="http://greath.example.com/2004/schemas/reservationDetails"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">

         <documentation>
                 This document describes the GreatH Retrieve Reservation
Details
                 Web service.
         </documentation>

         <types>

                 <xs:schema id="items"

targetNamespace="http://greath.example.com/2004/schemas/reservationItems">

                          <xs:element name="confirmationNumber" type="string"
/>
                          <xs:element name="checkInDate" type="date" />
                          <xs:element name="checkOutDate" type="date" />
                          <xs:element name="roomType" type="string" />
                          <xs:element name="smoking" type="boolean" />

                 </xs:schema>

                 <xs:schema id="details"

targetNamespace="http://greath.example.com/2004/schemas/reservationDetails"

xmlns:items="http://greath.example.com/2004/schemas/reservationItems">

                          <xs:import

namespace="http://greath.example.com/2004/schemas/reservationItems"
                                   schemaLocation="#items" />
```

```
                        <xs:element name="reservationDetails">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element
ref="items:confirmationNumber" />
                                                <xs:element
ref="items:checkInDate" />
                                                <xs:element
ref="items:checkOutDate" />
                                                <xs:element
ref="items:roomType" />
                                                <xs:element
ref="items:smoking" />
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:schema>

        </types>

        <interface name="retrieveDetailsInterface">

                <operation name="retrieve"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="#none" />
                        <output messageLabel="Out"
                                element="wdetails:reservationDetails" />
                </operation>

        </interface>

 </description>
```

4. Advanced Topics II: розширюваність і зумовлення розширень

4.1 Розширюваність

WSDL 2.0 забезпечує відкриту модель змісту, яка дозволяє XML елементам і атрибутам з інших (не WSDL 2.0) просторових імен XML, буде перемежовуватися в документі WSDL 2.0. Повне ім'я (в комплекті з простором імен URI) елемента розширення або атрибуту виступає як однозначне ім'я семантики, що розширюється. Простір імен URI розширеного елемента повинен бути dereferenceable на документ, що описує семантику, що розширює. На момент написання статті, не існує загальноприйнятого стандарту для документа, який повинен бути. Однак, W3C TAG обговорює питання (див. випуск namespaceDocument TAG-8) і може дати вказівки в деякій точці.

4.1.1 Факультативні Versus обов'язкові розширення

Розширення можуть бути або необхідними або факультативними. `Факультативне` розширення, яке клієнт може ігнорувати, виключно на свій розсуд, і сигналізує `wsld: Обов'язково = "помилкових"` або відсутність `wsld: обов'язковий` атрибуту (тому що воно за замовчуванням false). Таким чином, WSDL 2.0 процесор, діючи від імені клієнта, який стикається з невідомим розширенням факультативно може проігнорувати і продовжити процес WSDL 2.0 документа. *Тим не менш*, важливо підкреслити, що додаткове розширення тільки факультативне розширення для *клієнта,а не служба*. Служба має підтримувати всі необов'язкові розширення, і вимагати реклами у своєму документі WSDL 2.0.

Необхідне розширення, яке має бути підтримане і залучене клієнтами для того, щоб приступити до взаємодії належним чином, і сигналізує `wsld: required = "true"`. Якщо WSDL 2.0 процесор, діючи від імені клієнта, не визнає чи не підтримує, то він не може безпечно продовжити обробку документів WSDL 2.0. У більшості практичних випадків це може означати, що процесор буде вимагати ручного втручання для вирішення з розширенням. Наприклад, клієнт, розробник може вручну забезпечити здійснення необхідних для розширення WSDL 2.0 процесором.1 розширюваність

4.2 Визначення нових європарламентарії

Як ми згадували вже в **2.4.4.3 Message Exchange Patterns (Європарламентарії)**, хоча 8 європарламентаріїв визначається WSDL 2.0 призначені для покриття найбільш загальних випадків використання, існують ситуації, які вимагають нових євро парламентаріїв і вони повинні бути визначені. У цьому розділі ми пояснимо, як нові можуть бути визначені для вирішення спеціальних вимог бізнесу.
Після грандіозного успіху, Greath виявив, що це може призвести до радикального збільшення туристичного інтересу шляхом подання інформації про погодні умови, як для агентів бюро подорожей так і для широкої громадськості гастролей. This produced a challenge for the service implementers: how could this information be supplied to interested parties without requiring knowledge of web service technology specifically, and of computers generally? Питання полягало в бажанні забезпечити асинхронних оновлень для недосвідчених клієнтів без яких великі витрати на технічну підтримку. Було прийнято рішення створити стандартний список розсилки, і надати невелику крос-платформні веб-сервіс клієнта (фактично, абоненту), що може бути встановлене на будь-якому комп'ютері з POP або IMAP доступ до поштової скриньки. Поштові скринька, коли підписувалися на розсилки, можуть бути або оброблятися як "присвячений" (для служби погоди Greath; турагенти зробили це), або як "загального призначення" (в цьому випадку заява буде розглядати тільки ті листи, що містять Subject заголовки пов'язані зі службою). Це зажадало розробки обов'язковими до електронної пошти, яка виходить за рамки цього прикладу, але в результаті WSDL 2.0 був інший досить простий.
Примітка: Email обов'язково використовується тут на підтримку публікації / підписки, шляхом підтримки надійних виселення тільки МЕП, а також клієнт / сервер стилі In-Out використовувує для передплати та відписки.

Example 4-1. Weather Notification Service (Initial)

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
      targetNamespace="http://greath.example.com/2004/wsdl/weathSvc.wsdl"
      xmlns:tns="http://greath.example.com/2004/wsdl/weathSvc.wsdl"
      xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
      xmlns:email="http://www.example.com/webservices/email" >

    <types>
         .  .  .
    </types>

    <interface name="weatherInterface">
        <operation name="opSubscribeWeather"
                   pattern="http://www.w3.org/ns/wsdl/in-out">
            <input element=". . ." />
            <output element=". . ." />
        </operation>
        <operation name="opUnsubscribeWeather"
                   pattern="http://www.w3.org/ns/wsdl/in-out">
            <output element=". . ." />
            <input element=". . ." />
        </operation>
        <operation name="opNotifyWeather"
```

```
                    pattern="http://www.w3.org/ns/wsdl/robust-out-only">
            <output element=". . ." />
        </operation>
    </interface>

    <binding name="weatherMailingListBinding"
             interface="tns:weatherInterface
             type="http://www.w3.org/ns/wsdl/soap"
             wsoap:protocol="http://www.example.com/bindings/email">
              .  .  .
    </binding>

    <service name="weatherService"
             interface="tns:weatherInterface">
        <endpoint name="greatHWeatherList"
                  binding="tns:weatherMailingListBinding"
                  address="mailto:weather-owner@greath.example.com" />
    </service>

</description>
```

Примітка: у прикладі, messageLabels всіх вхідні і вихідні елементи були додатковими, оскільки вони не є необхідними для усунення неоднозначності (але зауважте, що наказ про вхід і вихід елемента не має істотного значення).
На жаль, програма незабаром захоплена з метою annoyment. Неодноразово в готелях менш сприятливий клімат, а також (урагани, смерчі) , клієнти були підписані на отримання повної інформації. Вони скаржилися Greath, на послуги дизайнера. Застосування інфраструктури відкритих ключів до вирішення проблеми була негайно відкинута як занадто складною і дуже важкою. Аналіз показав, що проблема була лишеперевіркою, що адреса, з проханням надати інформацію насправді хотіла цю інформацію. Таким чином, була визначена нова модель обміну повідомленнями.

4.2.1 Офіційно Challenge

Ця модель складається з двох або більше повідомлень в наступному порядку:
1. Повідомлення:
    - Про зазначення компонентів повідомлення етикету які помічені міткою "Запит" і напрямку "у регіоні"
    - Про отримані від деяких вузлів N1
2. Повідомлення:
    - Про зазначення компонентів повідомлення етикету які помічені міткою "виклик" і напрямку "аут"
    - спрямована деяким вузлом N2 (який може бути той же вузол, як № 1)
3. Необов'язкове повідомлення:
    - Про зазначення компонентів Label повідомлення етикету якого є "підтвердження" та напрямку "у регіоні"
    - Про отримання від вузла N2
4. Необов'язкове повідомлення:
    - Про зазначення компонентів Label повідомлення етикету якого є "відповідні заходи" і напрямку "аут"
    - Про відправлення на вузол N2

Ця модель використовує правило повідомлення ініціювання несправності.
Операції з використанням цієї схеми обміну повідомленнями є зразок власності зі значенням "http://www.example.com/webservices/meps/confirmed-challenge".
Після того, МООСа були визначені (та електронної пошти обов'язкових специфікацій

відповідним чином змінені, щоб вказати, що це підтримує MOOCa), служба була переглянута і перерозподілена. Тільки змінені операцій показано в уривку нижче.

Example 4-2. Weather Notification Service (Revised)

```
 <?xml version="1.0" encoding="utf-8" ?>
 <description xmlns="http://www.w3.org/ns/wsdl"
       targetNamespace="http://greath.example.com/2004/wsdl/weathSvc.wsdl"
       xmlns:tns="http://greath.example.com/2004/wsdl/weathSvc.wsdl"
       xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
       xmlns:email="http://www.example.com/webservices/email" >

     .  .  .

     <interface name="weatherInterface">
         <operation name="opSubscribeWeather"

pattern="http://www.example.com/webservices/meps/confirmed-challenge">
             <input messageLabel="Request" element=". . ." />
             <output messageLabel="Challenge" element=". . ." />
             <input messageLabel="Confirmation" element=". . ." />
             <output messageLabel="Response" element=". . ." />
         </operation>
         <operation name="opUnsubscribeWeather"

pattern="http://www.example.com/webservices/meps/confirmed-challenge">
             <output messageLabel="Challenge" element=". . ." />
             <output messageLabel="Response" element=". . ." />
             <input messageLabel="Confirmation" element=". . ." />
             <input messageLabel="Request" element=". . ." />
         </operation>
          .  .  .
     </interface>

      .  .  .

 </description>
```

Примітка: у другому прикладі, вхідний і вихідний приклад не в тій послідовності, в якій вони відбуваються в структурі, це свідчить про те, що послідовність не має істотного значення. Однак слід зазначити, що для цієї моделі, messageLabel атрибут необхідний на кожному вході і виході елемента.

4.3 RPC Стиль

Розділ **2.4.4.1 Атрибути операції** зазначив, що (за бажанням) атрибутом `стилю` інтерфейсів використовується, щоб вказати, які операції відповідають конкретно зумовленій операції стилю, або встановити обмеження. Дійсно, якщо бажаний `стиль` атрибута може містити список URIs, вказавши, що операція одночасно відповідає декільком стилям.

Операція стилю іменується URIs, щоб бути однозначною у той же час дозволяються нові стилі, які будуть визначені, не вимагаючи оновлень для мови WSDL 2.0. WSDL 2.0 Частина 2 [*WSDL 2.0 придатків*] визначає три такі стилі операції, один з яких є стиль RPC (RPC стилем).

Стиль *RPC* призначений для полегшення прив'язки мови програмування для WSDL 2.0 конструкцій. Це дозволяє WSDL 2.0 операції, яка буде легко відображатися на методі або функції підпису, таких як метод підпису в Java (TM) або C #. RPC Стиль обмежується операціями, які використовують In-Out (див. **2.4.4.3 розумінні Message Exchange Patterns**).

WSDL 2.0 документ використовує стиль RPC в інтерфейсі операції з першим визначенням операції у відповідності з усіма правилами RPC стилю, а потім встановивши, що `стиль` операції атрибут включає URI, який ідентифікує стиль RPC,

тим самим стверджуючи, що операція дійсно відповідає Стилю RPC. Ці правила дозволяють вхідним та вихідним схемам входи і виходи методу підпису. Грубо кажучи, елементи введення карток до вхідних параметрів, вихідних елементів, а також елементи, які з'являються як у схемах вхідних і вихідних повідомлень. WSDL 2.0 частини 2 розділу "НПК Стиль" надає повну інформацію про відображення правил і вимог.
Стиль RPC також забезпечує повний підпис призначення для відображення безпосередньо за допомогою `wrps: підпис` атрибуту визначено в WSDL 2.0 частини 2 розділу "wrps: підпис розширення". Це (за бажанням) розширення мови WSDL 2.0, значення якого визначає які вхідні і вихідні елементи повідомлень у методі підпису. Приклад нижче ілюструє, як RPC стиль може бути використаний для позначення підпису. Цей приклад є модифікованою версією послуги бронювання Greath. Зокрема, `інтерфейс` і `типи` розділів були змінені, щоб визначити і відповідати стилю RPC.

Example 4-3. Specifying RPC Style

```
  .  .  .
 <types>

   <xs:element name="checkAvailability">
     <xs:complexType>
       <xs:sequence>
           <xs:element  name="checkInDate" type="xs:date"/>
           <xs:element  name="checkOutDate" type="xs:date"/>
           <xs:element  name="roomType" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
   </xs:element>

   <xs:element name="checkAvailabilityResponse">
     <xs:complexType>
       <xs:sequence>
          <xs:element  name="roomType" type="xs:string"/>
          <xs:element  name="rateType" type="xs:string"/>
          <xs:element  name="rate" type="xs:double"/>
      </xs:sequence>
     </xs:complexType>
   </xs:element>
    .  .  .
 </types>

 <interface  name = "reservationInterface" >

   <operation name="checkAvailability"
         pattern="http://www.w3.org/ns/wsdl/in-out"
         style="http://www.w3.org/ns/wsdl/rpc"
         wrpc:signature=
           "checkInDate #in checkOutDate #in roomType #inout rateType #out
rate #return">
     <input messageLabel="In"
           element="tns:checkAvailability" />
     <output messageLabel="Out"
           element="tns:checkAvailabilityResponse" />

   </operation>
  .  .  .
 </interface>
  .  .  .
```

Зверніть увагу, що ім'я інтерфейсу операції "`checkAvailability`", так само, як локальна частина з QName вхідного елемента, "`tns: checkAvailability`". Це одна з вимог стилю RPC. Назва операції використовується як ім'я методу на мові обов'язкових, з урахуванням подальших обмежень на відображення конкретної

цільової мови програмування. У цьому випадку, назва методу буде "`checkAvailability`".
Зауважимо, що елементи `checkInDate`, `checkOutDate` є вхідними параметрами, однак `тип номера` елемента є In-Out параметр, які фігурують як елементи обох вхідних і вихідних елементів. Це свідчить про те, що застереження, система може змінити тип номера взалежності від наявності.
Служба бронювання також повертає тип ставки для бронювання, таких як "від стійки". Повертає значення методу позначається як "елемент курсу".
Виходячи з вартості `wrps: підпис` атрибуту метод підпису може бути отримано в порядку слідування параметрів. Приклад відображення наводиться нижче для Java (TM) мови. Цей приклад був створений за допомогою JAX RPC 1.1 [*JAX RPC 1.1*] для відображення простих типів Java типи і призначення INOUT і вихідних параметрів за допомогою Організатор класу.

Example 4-4. Sample Java(TM) Signature for RPC Style

```
public interface reservationInterface extends Remote{

   double checkAvailability(java.util.calendar checkInDate,
              java.util.calendar checkOutDate,
              StringHolder roomType,
              StringHolder rateType) throws RemoteException;
      .  .  .
 )
```

Мова програмування може також вказати, які несправності відображаються на мовних конструкцій та їх областей, таких, як виключення, але вони не є специфічними для стилю RPC.

5. Advanced Topics III: Різне

Цей розділ присвячений різним темам, які можуть підпадати під сферу WSDL 2.0, але може надати корисну довідку, яка може бути корисна при авторизації WSDL 2.0 документа або здійснення WSDL 2.0.

5.1 Дозволяючи легко Відправлення повідомлень

Бажано, щоб одержувач повідомлення мав можливість однозначно ідентифікувати тип повідомлення для правильної обробки. Можливість визначення типу повідомлень, як правило, використовується для диспетчерської мети в рамках реалізації веб-сервісів. Таким чином, WSDL автори рекомендували для розгляду питання про неоднозначність тиів повідомлень, коли вони розвивають свої послуги. Контексту, в якому веб-служби можуть бути розгорнуті грає важливу роль при виборі відповідного способу неоднозначность і визначення типу повідомлень. У типовому розгортанні Endpoint Address може приймати одна службу, яка описує сервісний елемент WSDL. У цьому випадку, коли використовується XSD, присвоєння унікального кваліфікованого імені глобальних заяв елемента в якості внеску в інтерфейс, який описує послугу буде достатньо для усунення неоднозначності типу повідомлень. Однак, коли в кінцевій точці адреси розміщено кілька послуг, по суті підтримує кілька описів WSDL, прагнення до неоднозначності типу повідомлення, слід розглядати в контексті всіх розгорнутих послуг, а не тільки в рамках єдиного інтерфейсу.
Як пояснюється в **2.4.4.1 Операція атрибутів**, коли XSD використовується як система типів, кілька спеціальних маркерів можуть бути використані як атрибути елементів. Однозначно що визначення типу повідомлення може стати дуже важким, якщо:

• Будь-який з цих елементів у рамках вхідного інтерфейсу має значення "# будь-якого" або

• Не більше одного з цих елементів введення (див. нижче) має значення "# None", або

• кваліфіковані імена глобальних елементів, зазначені в якості внеску елементів не є унікальними, коли розглядаються разом.

Якщо будь-який з трьох вищезгаданих випадків виникнення значення механізму можуть бути надані за допомогою розширення елемента (тобто елемент, який не в http://www.w3.org/ns/wsdl імені), що має WSDL: обов'язковий атрибут із значенням "істина". Семантика такого розширення означатиме елемент механізму однозначно identifing механізм, який відправнику повідомлення потрібий, щоб одержувач повідомлення однозначно визначив повідомлення.

Так, наприклад, WS-Addressing [*WS-Addressing*] специфікація передбачає таке значення механізму. Воно складається з елементів розширення, які можуть бути помічені як потрібно, і визначає необхідні [Action], вартість якого завжди присутня в сумісній доставці повідомлення. Значення дії може бути використане для неоднозначності повідомлення одержувачем, і існує також певний спосіб пов'язати дії до повідомлень в WS-Addressing специфікацій. Крім того, WS-Addressing також передбачає відповідні значення за замовчуванням дії, яка ідентифікує кожний тип повідомлення однозначно.

При використанні HTTP Binding, або за використання SOAP Binding відповідь ОЕТ, немає в конверті повідомлення-запит, і, отже, й інші механізми, ніж унікальні кваліфіковані імена глобальних заяв елементу, або заголовки таких як А3С: дій, повинні розглядатися. У цих випадках, (адреса) і () HTTP Location властивість можуть бути побудовані таким чином, щоб надати місця, які можуть бути співвіднесені з унікальними операціями. Наприклад, можна префікс HTTP Location з операцією імені, можна було б забезпечити, щоб частина HTTP Location попередньо була першою"(" характер бути унікальним в експлуатації).

5.2 Веб-служба версіями

WSDL 2.0 документ описує набір повідомлень, які веб-служби можуть відправляти й одержувати. По суті, вони описують мову для взаємодії з цією службою. Однак це можливо для веб-служби обміну повідомленнями інші крім тих, які описані в тому чи іншому документ WSDL 2.0. Часто це відбувається і, таким чином відбувається еволюція взаємодії мови.

Як краще керувати Evolution (версія) веб-систем є, на момент написання, предметом широкої дискусії. Однак, є три напрямки діяльності в рамках W3C, які мають безпосереднє відношення до опису версіями Web-сервісу:

• Технічна архітектура Group (TAG) опублікувало керівництво по розширюваності і версій форматів даних у веб-документі Архітектура [*Веб-архітектура*]. Існує також більш широкий діапазон проектів з пошуку та розширення версіями [*W3C TAG Висновок: Розширення і версіями Мови частина 1*]. Обидві ці роботи грунтуються на технічній записці про веб-архітектури: Extensible Мови [*WebArch: Extensible Мови*].

• XML Schema робоча група збирає серію прецедентів на схему управління версіями в рамках цієї схеми 1,1 діяльності. Див XML Schema версіями використання Справи [*XML Schema: Версія приклади використання*].

• Посібник з версіями XML мови XML Schema 1.1 [*Посібник з версіями XML Мови допомогою XML Schema 1.1*] ілюструє деякі методи для керування версіями XML Мови включення особливостей схеми XML 1.1 [*XML Schema 1.1*].

• Semantic Web Best Practices і розгортання робочох групи займається вивченням словників. Див [*SW VocabManagementNote*]

Хоча ця діяльність важливому в одному аспекті: що версія є важливим, але ми повинні передбачати і планувати зміни.

Проект з пошуку версій і розширюваності докладніше два основні підходи до версій:

• сумісна еволюція, і
• великого вибуху.

5.2.1 Сумісні еволюції

У *сумісний еволюції*, дизайнери, як очікується, обмежують зміни на ті, які є або вперед або назад сумісні, або обох:

Зворотна сумісність

Приймач поводиться правильно, якщо воно отримує повідомлення у *старій* версії взаємодії мови.

Переслати сумісні

Приймач поводиться правильно, якщо воно отримує повідомлення у *новій* версії взаємодії мови.

Оскільки веб-послуги і їхні клієнти відправляють й одержують повідомлення, ці поняття можуть застосовуватися для обох сторін. Проте, оскільки WSDL 2.0 сервіс-орієнтований, ми зосередимо увагу на справи служби еволюції.

Є три найважливіші області, в яких послуги описані в WSDL 2.0 розвиваються:

• Служба тепер підтримує додаткові сили. У сумісний еволюції, це має бути безпечно крім того, враховуючи, що додавання нових обов'язкових не повинно впливати на будь-яку існуючу взаємодію з використанням іншого транспорту.

• Інтерфейс підтримує нові операції. Знову ж таки, в сумісній еволюції звичайно це безпечно, враховуючи, що додавання додаткових операцій абстрактного інтерфейсу не повинна впливати на будь-які існуючі взаємодії.

• Повідомлення можуть включати додаткові дані. Який зміст повідомлення може змінюватися в описі залежить великою мірою від типу системи яка використовується для опису вмісту повідомлень. RelaxNG [*RELAX NG*] має хорошу підтримку для опису словників, яка ігнорує невідомі XML, так само як і OWL / RDF. XML Schema 1.0 має обмежену підтримку для розширення Опис повідомлення через `xs: будь-який` і `xs:anyAttribute` конструкцій. XML Schema 1.1 була призначена для забезпечення "змін, необхідних для забезпечення більш ефективної підтримки версій схем", і передбачається, що це може включати поліпшену підтримку для більш "відкритого контенту", і тому більш ефективної підтримки сумісної еволюції повідомлень.

• протокол використовується для обміну повідомленнями може надавати механізми для обміну даними поза тілами повідомлення. У разі SOAP, WSDL 2.0 обов'язкового дає можливість застосування для опису даних, що підлягають обміну, як заголовки. Модель обробки SOAP має дуже хорошу модель розширюваності з невідомим заголовком ігнорує приймач за замовчуванням. Існує також механізм, за допомогою якого заголовки, які необхідні як частина несумісні зміни можуть бути помічені прапором 'MustUnderstand. Переходячи додаткові предмети, як заголовки можуть бути єдиним способом розвиватися повідомленнями з фіксованими даними.

5.2.2 Великого Вибуху

Вибух великий підходу до версій простіше в даний час представляють у WSDL 2.0. При такому підході будь-які зміни в WSDL 2.0 Документ передбачають зміну імен документа, зміни в інтерфейс на увазі нові імена інтерфейсів і змін до змісту повідомлення повідомляється за допомогою нових імен повідомлення. Такий підхід має особливі переваги, коли агент може швидко сказати, якщо послуга не змінилася простим порівнянням імен значення.

5.2.3 Розвиток служби

Сумісні зміни набагато легше керовані, ніж несумісні, з них:

• Зі зміною сумісної послуги потрібна тільки підтримка останньої версії служби. Клієнт може продовжувати користуватися послугою, пристосуватися до нової версії інтерфейсу опису в той час на свій вибір.

• З несумісних змін, клієнт отримує нову версію опису інтерфейсу і, як очікується, щоб пристосуватися до нового інтерфейсу до припинення старим інтерфейсом. Будь-яка послуга повинна продовжувати підтримувати обидві версії інтерфейсу в ході передачі періоду, або службою та клієнтами узгоджуються зі змінами в той же час. Альтернативою є для клієнта, буде тривати до виявлення помилки, після чого він використовує нову версію інтерфейсу.

5.2.4 Комбіновані підходи

Доцільно об'єднати "сумісну еволюцію" і "великий вибух" в підходах до різних способів. Наприклад, простіі імен може бути змінений при зміні повідомлення описів, але імена можуть залишатися такими ж, коли додаються нові операції.
Хоча великий Bang підхід є в даний час легший за все здійснити у WSDL 2.0, це може призвести до великої кількості клонованих інтерфейсів, стає важко керувати. Врешті-решт, вибір версіями стратегію для веб-сервісів WSDL описано в 2.0 в якості вправи для читача.

5.2.5 Приклад версіями і розширення служби

5.2.5.1 Додаткові елементи, додані до контент

Наступний приклад демонструє, як зміст може бути розширений за рахунок додаткового змісту. Службу бронювання змінено на більш нову версію, яка може прийняти факультативну кількість гостей . Послуга існуючого клієнта має можливість використовувати послугу. Автор додає елемент у схемі в якості додаткового елементу.

Example 5-1. XML Schema with Optional Elements

```
<xs:complexType name="tCheckAvailability">
    <xs:sequence>
      <xs:element  name="checkInDate" type="xs:date"/>
      <xs:element  name="checkOutDate" type="xs:date"/>
      <xs:element  name="roomType" type="xs:string"/>
      <xs:element  name="numberOfGuests" type="xs:integer" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax"/>
    </xs:sequence>
</xs:complexType>
```

Автор має можливість вибору збереження імен або ж використати інші імена для додаткового вмісту і наявного змісту. У цьому випадку, це сумісна зміна і автор вирішить зберегти ці ж імена. Це дозволить існуючим клієнтам взаємодіяти з новою послугою.

.

5.2.5.2 Додаткові елементи додаються в заголовок

Інший варіант полягає в розширенні як заголовок блоку. Це досягається шляхом визначення елементів для розширення і додавання заголовка елемента, який посилається на елемент в обов'язкову операцію введення.

Example 5-2. Additional optional elements added to a SOAP header

```
<xs:element name="NumberOfGuests" type="tNumberOfGuests"/>
<xs:complexType name="tNumberOfGuests">
    <xs:sequence>
        <xs:element  name="numberOfGuests" type="xs:integer" minOccurs="0"/>
    </xs:sequence>
```

```
</xs:complexType>

<binding name="reservationSOAPBinding"
    interface="tns:reservationInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <operation ref="tns:opCheckAvailability">
       <input>
         <wsoap:header element="tns:NumberOfGuests"/>
       </input>
    </operation>
 ...
</binding>
```

Це також можливо для заголовка: MustUnderstand набір до істини. HTTP Binding має схожу функціональність хоча і без MustUnderstand атрибуту.

5.2.5.3 Додаткові обов'язкові елементи в змісті

Наступний приклад демонструє розширення з додатковим змістом. Бронювання послуг вимагає числа гостей параметрів. Постачальник послуг хоче, щоб існуючі клієнти змогли користуватися послугою. Автор додає елемент у схемі як обов'язковий елемент.

Example 5-3. Additional Mandatory Elements in Content

```
<xs:complexType name="tCheckAvailabilityV2">
    <xs:sequence>
    <xs:element  name="checkInDate" type="xs:date"/>
    <xs:element  name="checkOutDate" type="xs:date"/>
    <xs:element  name="roomType" type="xs:string"/>
    <xs:element  name="numberOfGuests" type="xs:integer"/>
    <xs:any namespace="##other" processContents="lax"/>
    </xs:sequence>
</xs:complexType>
```

Автор має можливість вибору збереження імен або ж використати інші імена для додаткового вмісту і наявного змісту. У цьому випадку, це несумісні зміни і автор вирішує використовувати нове ім'я, але той же простір імен. Цей тип використовується потім в інтерфейсі операції, а потім обов'язкову службу і кінцеві точки.

5.2.5.4 Додаткові операції додано до інтерфейсу

Розділ **2.4.2 успадкування інтерфейсу** показаний ще один тип керування версіями або продовження, де reservationInterface розширює MessageLogInterface. За визначенням успадкування інтерфейсу, клієнт розуміє, що тільки MessageLogInterface буде продовжувати працювати з reservationInterface, і що він має зворотну сумісність.

5.2.5.5 Додаткову обов'язкову операцію додано до інтерфейсу

Часто обов'язкові операції додані до інтерфейсу. Готельний сервіс вирішує додати операцію проти застереження сервісу, який є підтвердженням. Готельний сервіс вимагає, щоб всі клієнти перейшли на новий інтерфейс для використання сервісу. Вони мають різні варіанти з вказівкою, що є застарілим старим інтерфейсом.
За визначенням успадкування інтерфейсу, вони не можуть використовувати успадкування інтерфейсу для визначення розширення.

Example 5-4. Additional Mandatory Operation Added to the Interface

```
<interface name="reservationWithConfirmation" extends="cc:creditCardFaults">
     ...
    <operation name="makeReservation">
        <input messageLabel="In" element="ghns:makeReservation" />
        <output messageLabel="Out" element="ghns:makeReservationResponse" />
        <outfault ref="invalidDataFault" messageLabel="Out" />
        <outfault ref="cc:cancelledCreditCard" messageLabel="Out" />
        <outfault ref="cc:expiredCreditCard" messageLabel="Out" />
        <outfault ref="cc:invalidCreditCardNumber" messageLabel="Out" />
        <outfault ref="cc:invalidExpirationDate" messageLabel="Out" />
    </operation>
    <operation name="confirmReservation">
        <input messageLabel="In" element="ghns:makeReservationResponse" />
        <output messageLabel="Out" element="ghns:confirmReservationResponse"
/>
        <outfault ref="expiredReservation" messageLabel="Out" />
    </operation>
</interface>
```

Цей інтерфейс не може бути пов'язаний і розгорнутий на існуючому URI і вказаний несумісності, так як послуга буде як і раніше makeReservation прохання. Зміна імені інтерфейсу з застереженням, щоб reservationWithConfirmation або зміну назви операції від makeReservation до makeReservationV2 не впливала на повідомлення обміну. Тому вона не може бути використана в якості механізму для індикації несумісності. Щоб показати несумісність, зміни повинні бути зроблені на те, що відображається в повідомленні. Для більш SOAP HTTP запиту, список приблизно URI, дія HTTP заголовок або зміст повідомлення.

5.2.5.6 Зазначення Несумісності шляхом зміни Endpoint URI

Щоб показати несумісність, Урі з готелю точки може бути змінена і відправленна повідомлення на повернення старої Endpoint Fault.

5.2.5.7 Зазначення несумісності шляхом зміни SOAP дії

SOAP дія може бути встановлена для makeReservation прохання, і що робить його відмінним від попередніх версій необхідно вказати несумісності.

Example 5-5. Indicating Incompatibility by changing the SOAP Action

```
<binding name="reservationSOAPBinding"
        interface="tns:reservationInterface"
        type="http://www.w3.org/ns/wsdl/soap"
        wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
    <operation ref="tns:makeReservation"
        wsoap:action="tns:makeReservationV2"/>
     .  .  .
```

Зауважимо, що цей механізм застосовується для кожної обов'язкової основи. SOAP HTTP Binding забезпечує налаштування дій, але й інші прив'язки не можуть забезпечити такий механізм.

5.2.5.8 Зазначення несумісності шляхом зміни елементів змісту

Імена або назви makeReservation елемент можуть бути змінені, а потім інтерфейс і прив'язка змінилася. Щоб показати несумісність, запити, використовуючи старі QName makeReservation. Новий інтерфейс, за зміненим makeReservation, це:

Example 5-6. Indicating incompatibility by changing the element content

```
<xs:element name="ghns2:makeReservation" type="ghns:tmakeReservation"/>
```

```
<interface .  . .>

   <operation name="makeReservation">

       <input messageLabel="In" element="ghns2:makeReservation" />

</interface>
```

Обов'язкові та кінцеві точки служби не вимагають ніяких змін.
Нарешті, ця служба може також надавати інтерфейс для ghns: makeReservation, що тільки повертає помилка.

5.3 Опису веб-служб повідомлень, що посилаються на інші веб-послуги

Посилання є однією з визначальних характеристик Web. Можливість переходу з однієї веб-сторінки до іншої надзвичайно корисна. Тому цілком природно, щоб застосувати цю можливість на веб-сервісі. У цьому розділі описуються посилання на кінцях і послуг.
Посилання на кінцевій точці елемента або атрибута, який містить адресу кінцевої точки веб-служби. *Посилання на послуги* елемента або атрибута, який містить одну або кілька посилань на кінцях служби. Якщо інтерфейс може корисно додати цю інформацію в WSDL 2.0 документ, який описує веб-служби. Це досягається за допомогою `wsdlx: інтерфейс` або `wsdlx: обов'язковий` атрибут компонента XML схеми, який визначає повідомлення.
Можна запитати з від веб-архітектурної точки зору, тому нічого більше URI необхідно буде посилатися на веб-служби. Дійсно, посилання на службу робить використання одного або більше URI, щоб вказати кінцеву точку адрес служби. Однак, він також може включати додаткові метадані про цю службу, такі як WSDL 2.0 і обов'язковим, що служба підтримує.
Перелік послуг та кінцеві точки будуть проілюстровані на прикладі розширення Greath вже обговорювали.

5.3.1 Резервування інформації Web-служби

При розробці веб-додатків, природно дати кожному важливу концепцію URI. У системі бронювання готелів Greath, важливі концепції є застереженнями, тому ми почнемо наш дизайн шляхом присвоєння URI для кожного застереження. Оскільки кожне застереження має свій унікальний номер підтвердження, наприклад OMX736, ми створюємо URI для кожного застереження шляхом додавання номеру підтвердження на базу URI, наприклад http://greath.example.com/2004/reservation/OMX736. Це URI буде кінцева точка адреса для бронювання інформації Web-служби, яка може одержувати й оновлювати стан бронювання. Приклад 5-7 показаний формат застереження докладно.
Example 5-7. Detail for Reservation OMX736

```
 <? XML Version = "1.0" кодування = "UTF-8"?
<reservationDetails
        xmlns="http://greath.example.com/2004/schemas/reservationDetails">

        <confirmationNumber>OMX736</confirmationNumber>
        <checkInDate>2005-06-01</checkInDate>
        <checkOutDate>2005-06-03</checkOutDate>
        <roomType>single</roomType>
        <smoking>false</smoking>

</reservationDetails>
```

Бронювання інформації Web-сервісу забезпечує операції з пошуку та оновлення докладно застереження. Приклади 5-8 наведені описи для цієї веб-служби. Зверніть увагу, що не існує `послуги` елемента у цьому описі, оскільки безліч застережень є динамічними. Замість того, кінці застережень, будуть повернуті шляхом запиту бронювання списку веб-служби.

Example 5-8. The Reservation Details Web Service Description: reservationDetails.wsdl

```
<?xml version="1.0" encoding="utf-8" ?>
<description
        xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/reservationDetails"
        xmlns:tns="http://greath.example.com/2004/services/reservationDetails"

xmlns:wdetails="http://greath.example.com/2004/schemas/reservationDetails"
        xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <documentation>
                This document describes the GreatH Reservation Details Web
                services. Use these services to retrieve or update reservation
                details. Each reservation has its own service and endpoint. To
                obtain the reference for a reservation service, make a request
to
                the GreatH Reservation List Web service. See
                reservationList.wsdl for a description of the Reservation List
                Web service.
        </documentation>

        <types>
                <xs:import

namespace="http://greath.example.com/2004/schemas/reservationDetails"
                        schemaLocation="reservationDetails.xsd" />
        </types>

        <interface name="reservationDetailsInterface">

                <operation name="retrieve"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="#none" />
                        <output messageLabel="Out"
                                element="wdetails:reservationDetails" />
                </operation>

                <operation name="update"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In"
                                element="wdetails:reservationDetails" />
                        <output messageLabel="Out"
                                element="wdetails:reservationDetails" />
                </operation>

        </interface>

        <binding name="reservationDetailsSOAPBinding"
                interface="tns:reservationDetailsInterface"
                type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

                <operation ref="tns:retrieve"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />
```

```
                <operation ref="tns:update"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />

        </binding>

</description>
```

Example 5-9 shows the XML schema elements that are used in this Web service.

Example 5-9. The Reservation Details Web Service XML Schema: reservationDetails.xsd

```
  <? XML Version = "1.0" кодування = "UTF-8"?
 <schema xmlns="http://www.w3.org/2001/XMLSchema"
         elementFormDefault="qualified"

targetNamespace="http://greath.example.com/2004/schemas/reservationDetails"
         xmlns:tns="http://greath.example.com/2004/schemas/reservationDetails"

xmlns:wdetails="http://greath.example.com/2004/services/reservationDetails"
         xmlns:wsdli="http://www.w3.org/ns/wsdl-instance"
         xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"

wsdli:wsdlLocation="http://greath.example.com/2004/services/reservationDetails
reservationDetails.wsdl">

         <element name="confirmationNumber" type="string" />

         <element name="checkInDate" type="date" />

         <element name="checkOutDate" type="date" />

         <element name="reservationDetails">
                 <complexType>
                         <sequence>
                                 <element ref="tns:confirmationNumber" />
                                 <element ref="tns:checkInDate" />
                                 <element ref="tns:checkOutDate" />
                                 <element name="roomType" type="string" />
                                 <element name="smoking" type="boolean" />
                         </sequence>
                 </complexType>
         </element>

         <simpleType name="reservationDetailsSOAPEndpointType"
wsdlx:binding="wdetails:reservationDetailsSOAPBinding">
                 <restriction base="anyURI"/>
         </simpleType>

         <element name="reservationDetailsSOAPEndpoint"
type="tns:reservationDetailsSOAPEndpointType" />

         <element name="reservationDetailsService">
                 <annotation>
                         <documentation>
                                 This element contains references to the
Reservation
                                 Details Web Service endpoints for this
reservation.
                         </documentation>
                 </annotation>
                 <complexType>
                         <sequence>
                                 <element name="soap"
type="tns:reservationDetailsSOAPEndpointType"/>
                                 <element name="secure-soap"
type="tns:reservationDetailsSOAPEndpointType"/>
                         </sequence>
```

```
                </complexType>
        </element>

 </schema>
```

Ця схема містить XML звичайні визначення елементів, які наводяться в повідомленні веб-служби. Наприклад, `reservationDetails` елемент використовується в посланнях `отримування` та `оновлювання` операцій. Крім того, схема визначає просту `reservationDetailsSOAPEndpointType` типу, який базується на `xs:anyURI` та анотації `wsdlx: обов'язкові = "wdetails: reservationDetailsSOAPBinding"`, яка означає, що URI є адресою бронювання інформації Web-послуги кінцевої точки, яка реалізує `wdetails: reservationDetailsSOAPBinding` . Зверніть увагу, що `wsdli: wsdlLocation` атрибут використовується для визначення місця розташування WSDL 2.0 документу, який визначає `wdetails: reservationDetailsSOAPBinding` обов'язковим. Це простий тип використовування для визначення `reservationDetailsSOAPEndpoint` елементу, який буде використовуватися в списку послуг бронювання.

5.3.2 Список бронювання Веб-службами

Оскільки безліч застережень зроблені і скасовані, то застереження не описані у фіксованому документі WSDL 2.0. Замість цього вони повертаються у вигляді відповідей на запити, зроблені на службу бронювання списком Інтернет. Адреса кінцевої точки для служби бронювання списку буде http://greath.example.com/2004/reservationList.
Приклад 5-10 показує формат відповіді від служби списку бронювання.

Example 5-10. Response from the Reservation List Web Service

```
  <? XML Version = "1.0" кодування = "UTF-8"?
 <reservationList
        xmlns="http://greath.example.com/2004/schemas/reservationList"

xmlns:details="http://greath.example.com/2004/schemas/reservationDetails">

        <reservation>

<details:confirmationNumber>HSG635</details:confirmationNumber>
                <details:checkInDate>2005-06-27</details:checkInDate>
                <details:checkOutDate>2005-06-28</details:checkOutDate>
                <details:reservationDetailsSOAPEndpoint>
                        http://greath.example.com/2004/reservation/HSG635
                </details:reservationDetailsSOAPEndpoint>
        </reservation>

        <reservation>

<details:confirmationNumber>OMX736</details:confirmationNumber>
                <details:checkInDate>2005-06-01</details:checkInDate>
                <details:checkOutDate>2005-06-03</details:checkOutDate>
                <details:reservationDetailsSOAPEndpoint>
                        http://greath.example.com/2004/reservation/OMX736
                </details:reservationDetailsSOAPEndpoint>
        </reservation>

        <reservation>

<details:confirmationNumber>WUH663</details:confirmationNumber>
                <details:checkInDate>2005-06-11</details:checkInDate>
                <details:checkOutDate>2005-06-15</details:checkOutDate>
                <details:reservationDetailsSOAPEndpoint>
                        http://greath.example.com/2004/reservation/WUH663
                </details:reservationDetailsSOAPEndpoint>
        </reservation>
```

```
 </reservationList>
```

Тут `<details:reservationDetailsSOAPEndpoint>` елементи містять посилання на бронювання інформації Web-послуги для кінцевих точок застережень HSG635, OMX736 і WUH663.
Приклад 5-11 показує приклад опису послуги бронювання списку Інтернет. Зауважимо, що вона містить операції, щоб дістатись до списку і для запиту списку застережень підтвердження номеру, дати заїзду та дати виїзду. У кожному випадку операція повертає список застережень.

Example 5-11. The Reservation List Web Service Description: reservationList.wsdl

```
<?xml version="1.0" encoding="utf-8" ?>
<description
        xmlns="http://www.w3.org/ns/wsdl"

targetNamespace="http://greath.example.com/2004/services/reservationList"
        xmlns:tns="http://greath.example.com/2004/services/reservationList"

xmlns:details="http://greath.example.com/2004/schemas/reservationDetails"
        xmlns:list="http://greath.example.com/2004/schemas/reservationList"
        xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <documentation>
                This document describes the GreatH Reservation List Web
                services. Use this service to retrieve lists of reservations
                based on a variety of search criteria.
        </documentation>

        <types>
                <xs:import

namespace="http://greath.example.com/2004/schemas/reservationDetails"
                        schemaLocation="reservationDetails.xsd" />
                <xs:import

namespace="http://greath.example.com/2004/schemas/reservationList"
                        schemaLocation="reservationList.xsd" />
        </types>

        <interface name="reservationListInterface">

                <operation name="retrieve"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="#none" />
                        <output messageLabel="Out"
element="list:reservationList" />
                </operation>

                <operation name="retrieveByConfirmationNumber"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In"
                                element="details:confirmationNumber" />
                        <output messageLabel="Out"
element="list:reservationList" />
                </operation>

                <operation name="retrieveByCheckInDate"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In" element="details:checkInDate"
/>
                        <output messageLabel="Out"
element="list:reservationList" />
                </operation>

                <operation name="retrieveByCheckOutDate"
```

```
                        pattern="http://www.w3.org/ns/wsdl/in-out">
                        <input messageLabel="In"
element="details:checkOutDate" />
                        <output messageLabel="Out"
element="list:reservationList" />
                </operation>

        </interface>

        <binding name="reservationListSOAPBinding"
                interface="tns:reservationListInterface"
                type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

                <operation ref="tns:retrieve"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />

                <operation ref="tns:retrieveByConfirmationNumber"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />

                <operation ref="tns:retrieveByCheckInDate"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />

                <operation ref="tns:retrieveByCheckOutDate"
                        wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />

        </binding>

        <service name="reservationListService"
                interface="tns:reservationListInterface">

                <endpoint name="reservationListEndpoint"
                        binding="tns:reservationListSOAPBinding"

address="http://greath.example.com/2004/reservationList" />

        </service>

</description>
```

Example 5-12 shows the schema for the messages used in the Reservation List Web service.

Example 5-12. The Reservation List Schema: reservationList.xsd

```
  <? XML Version = "1.0" кодування = "UTF-8"?
 <schema xmlns="http://www.w3.org/2001/XMLSchema"
         elementFormDefault="qualified"

targetNamespace="http://greath.example.com/2004/schemas/reservationList"
         xmlns:tns="http://greath.example.com/2004/schemas/reservationList"

xmlns:details="http://greath.example.com/2004/schemas/reservationDetails"
         xmlns:wsdli="http://www.w3.org/ns/wsdl-instance">

         <import
                 namespace="http://www.w3.org/ns/wsdl-instance" />

         <import

namespace="http://greath.example.com/2004/schemas/reservationDetails"
                 schemaLocation="reservationDetails.xsd" />
```

```
        <element name="reservation">
                <annotation>
                        <documentation>
                                A reservation contains the confirmation
number, check-in
                                and check-out dates, and a reference to a
Reservation
                                Details Web service.
                        </documentation>
                </annotation>
                <complexType>
                        <sequence>
                                <element ref="details:confirmationNumber" />
                                <element ref="details:checkInDate" />
                                <element ref="details:checkOutDate" />
                                <element
ref="details:reservationDetailsSOAPEndpoint" />
                        </sequence>
                </complexType>
        </element>

        <element name="reservationList">
                <annotation>
                        <documentation>
                                A reservation list contains a sequence of
zero or more
                                reservations.
                        </documentation>
                </annotation>
                <complexType>
                        <sequence>
                                <element ref="tns:reservation" minOccurs="0"
                                        maxOccurs="unbounded">
                                </element>
                        </sequence>
                        <attribute ref="wsdli:wsdlLocation" />
                </complexType>
        </element>

 </schema>
```

У попередньому прикладі, була одна кінцева точка, пов'язана з кожною послугою бронювання в докладному Інтернеті. Нехай Greath готель вирішить надати другу, безпечну кінцеву точку. У даному випадку посилання на послуги будуть використовуватися для збору разом кожного бронювання. ReservationDetails.xsd схема визначає `reservationDetailsService` елементи для цієї мети. Він містить `вкладені` елементи і `безпечні`, які кожен з типів `reservationDetailsSOAPEndpointType` і тому містить адресу кінцевої точки, яка реалізує `wdetails:reservationDetailsSOAPBinding` обов'язковим.
Приклади 5-13 показує приклад повідомлення, яке містить посилання на цю послугу замовлення HGS635. Зверніть увагу, що послуга складається з двох кінців, один з яких забезпечує безпечний доступ до послуги бронювання інформації Web.

Example 5-13. A Reference to the Reservation Details Web Service

```
  <? XML Version = "1.0" кодування = "UTF-8"?
 <details:reservationDetailsService

xmlns:details="http://greath.example.com/2004/schemas/reservationDetails"

        <details:soap>
                http://greath.example.com/2004/reservation/HSG635
        </details:soap>

        <details:secure-soap>
                https://greath.example.com/2004/reservation/HSG635
```

```
        </details:secure-soap>

 </details:reservationDetailsService>
```

5.3.3 Резервування інформації Web-служби за допомогою HTTP передачі

Даний розділ являє собою варіацію на прикладі у розділі **5.3.1 Резервування інформації Web-служби**. Вона ілюструє використання операції передачі HTTP, GET і PUT, для отримання та оновлення Greath бронювання готелів деталей методом передачі Representational держави (REST) архітектурного стилю описаного Рой Філдінг [REST]. Так як кожне замовлення в нашому прикладі буде мати різні URI, бронювання інформація Web-послуги може бути запропонована за допомогою HTTP GET і HTTP PUT. Обов'язкові будуть змінені в такий спосіб:

Example 5-14. Reservation Details Web Service Using HTTP Transfer

```
  .  .  .
 <binding name="reservationDetailsHTTPBinding"
         type="http://www.w3.org/ns/wsdl/http"
     interface="tns:reservationDetailsInterface" >

         <operation ref="tns:retrieve"
             whttp:method="GET" />

         <operation ref="tns:update"
             whttp:method="PUT" />

 </binding>
  .  .  .
```

Як приклад у розділі **5.3.1 Резервування інформація Web-служби**, служби і кінцевих елементів не передбачено, оскільки список веб-служби бронювання забезпечує кінцеві точки.

5.3.4 Бронювання списку веб-служби за допомогою HTTP GET

У цьому розділі продовжуєтьсяч REST-стиль наприклад **5.3.3 Резервування інформації Web-служби за допомогою HTTP передачі** шляхом зміни. Наприклад **5.3.2 Список Бронювання веб-служби** для використання HTTP GET.
Версія SOAP списку веб-служби бронювання вище пропонує чотири різних пошукових операції. Вони також можуть бути висловлені різними параметрами в URI використовуючи HTTP GET:

Example 5-15. Reservation List Web Service Using HTTP GET

```
  .  .  .
 <binding name="reservationListHTTPBinding"
         type="http://www.w3.org/ns/wsdl/http"
     interface="tns:reservationListInterface"
     whttp:methodDefault="GET">

   <operation ref="tns:retrieve"
       whttp:location="" />

   <operation ref="tns:retrieveByConfirmationNumber"

whttp:location="reservationList/ConfirmationNumber/{confirmationNumber/}" />

   <operation ref="tns:retrieveByCheckInDate"
       whttp:location="reservationList/CheckInDate/{checkInDate/}" />

   <operation ref="tns:retrieveByCheckOutDate"
       whttp:location="reservationList/CheckOutDate/{checkOutDate/}" />
 </binding>
  .  .  .
```

```
<service .  .  .  >

  <endpoint name="reservationListEndpoint"
    binding="tns:reservationListHTTPBinding"
    address="http://greath.example.com/2004/reservationList" />
 .  .  .
</service>
 .  .  .
```

Пошук по введенню номера URI буде виглядати так:
`http://greath.example.com/2004/reservationList/ConfirmationNumber/HSG635`.
Крім того, один тип запиту може бути наданий. Цей запит типу є послідовність додаткових елементів. Всі предмети в тій послідовності, серіалізуются в рядок запиту URI. Послідовність запиту для будь-якого з ConfirmationNumber, checkInDate, checkOutDate буде виглядати наступним чином:

Example 5-16. Query Sequence Using a Single Query Type

```
<element name="reservationQuery">
  <annotation>
    <documentation>
      A reservation contains the confirmation number, check-in
      and check-out dates, and a reference to a Reservation
      Details Web service.
    </documentation>
  </annotation>
  <complexType>
    <sequence>
      <element ref="details:confirmationNumber" minOccurs="0"/>
      <element ref="details:checkInDate" minOccurs="0"/>/>
      <element ref="details:checkOutDate" minOccurs="0"/>/>
    </sequence>
    </sequence>
  </complexType>
</element>
```

WSDL 2.0, яка пропонує тип як параметр, який буде виглядати наступним чином:
Example 5-17. WSDL 2.0 for Using a Single Query Type

```
 .  .  .
<interface name="reservationListInterfaceWithQuery">

  <operation name="retrieveByReservationQuery"
                pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In"
        element="details:ReservationQuery" />
    <output messageLabel="Out"
        element="list:reservationList" />
  </operation>

</interface>

<binding name="reservationListQueryHTTPBinding"
        type="http://www.w3.org/ns/wsdl/http"
    interface="tns:reservationListInterfaceWithQuery"
    whttp:methodDefault="GET">

  <operation ref="tns:retrieveByReservationQuery"
      whttp:location="reservationList/{ReservationQuery}}" />

</binding>

 .  .  .
  <endpoint name="reservationListEndpoint"
      binding="tns:reservationListHTTPBinding"
      address="http://greath.example.com/2004/reservationList" />
 .  .  .
```

Різні URI, будуть:
`http://greath.example.com/2004/reservationList/ReservationQuery?confirmationNumber=HSG635`
`http://greath.example.com/2004/reservationList/ReservationQuery?checkInDate=06-06-05.`
Важливо відзначити, що використання серіалізації URI може привести до дуже гнучких запитів і кількох операцій.

5.4 Декілька інтерфейсів для тієї ж служби

Припустимо, що веб-служба хотіла б вплинути на два різні інтерфейси: інтерфейс клієнта для своїх постійних користувачів, і інтерфейс для управління своїм оператором. `wsdl:служба` визначає лише однин wsdl: інтерфейс, тому для досягнення бажаного ефекту послуг необхідно вказати, відносини між двома службами. Як постачальник послуг свідчить про взаємозв'язок між послугами? Можливі стратегії включають в себе:
• **Заявляємо, обидва інтерфейси в тому ж wsdl: описі елементів**. Хоча WSDL 2.0 не приписує ніякого значення той факт, що дві WSDL: послуги, заявлені у рамках того ж WSDL: опис, додаток або інструментарій може інтерпретувати це в тому сенсі, що вони пов'язані в деякому роді.
• **Заявляємо, обидва інтерфейси в тому ж wsdlL: TargetNamespace.** Знову ж таки, хоч і WSDL 2.0 не приписує ніякого значення той факт, що дві WSDL: послуги, заявлені у рамках того ж WSDL: TargetNamespace, додаток або інструментарій може інтерпретувати це в тому сенсі, що вони пов'язані в деякому роді.
• **Додати розширенням WSDL 2.0**, яке пов'язує всі служби, пов'язані з подібним чином. WSDL 2.0 "з відкритою моделлю змісту дозволяє розширити елементи з інших імен в документі WSDL 2.0.
• **оголосити їх в зовсім окремий WSDL 2.0 документ, і використати ту саму адресу кінцевої точки для обох сторін.** Тобто, оголосити `wsdl: інтерфейс` і `wsdl: послуги` для клієнтів інтерфейсу в одному документі WSDL 2.0 і `wsdl: інтерфейс` і `wsdl: послуги` для інтерфейсу управління в іншому документі WSDL 2.0, але і використовувати ту саму адресу кінцевої точки для обох сторін. (Під "різними WSDL 2.0 документа" ми маємо на увазі, що обидва документи ніколи не включати або імпортувати в той же WSDL 2.0 компонент опису.) Хоча цей підхід може в деяких випадках, означає, що ту саму адресу кінцевої точки будуть використовуватися для двох різних цілей. Крім того, воно йде врозріз з веб-архітектурним принципом, згідно з яким різні URI, повинні бути використані для ідентифікації різних веб-ресурсів. (Див. веб-архітектура [*Веб-архітектура*] розділ зіткнення URI collision.)
• **Використання успадкування об'єднання клієнта і інтерфейсу** управління в єдину, велику wsdl:інтерфейс. Звичайно, це знижує модуль і означає, що інтерфейс управління стає піддаватися клієнту, що не є добрим.
Майте на увазі, що, оскільки вищезгадані кроком стратегії за межами мови WSDL 2.0 передбачають (і отже, не схвалили ні заборонена WSDL специфікації 2.0) WSDL 2.0 специфікації не можуть визначити та стандартизувати їх семантику.
Бажання висловити відносини між службами, також має відношення до версій веб-сервісу, обговоримо нижче.

5.5 Карта для RDF і Semantic Web

WSDL 2.0 мова призначена в першу чергу для синтаксису XML. Хоча майже всі розуміють, у нього є кілька питань:
• Можливість складати два XML-документів в один залежить від мови цих документів. WSDL 2.0 не дозволяє веб-службам описи у різних targetNamespaces бути об'єднані в один (фізичний) XML документ.
• Можливість продовжити XML мови з іншими мовами XML залежить від мови знову.

WSDL 2.0має надзвичайно можливі розширення, але значення кожного розширення в WSDL 2.0, повинне бути визначене в явному вигляді. Вважаючи XMI (XML формат UML) в WSDL 2.0 документ може мати інше значення, поклавши його в документ XHTML. Тому на основі XML розширення має дуже високу вартість, якщо займатися багатьма мовами.
• Аналогічним чином, розширення іншої мови XML з частинами WSDL 2.0, повинно бути визначене на всіх можливих напрямках. Вважаючи WSDL 2.0 інтерфейс елемента в реєстрі UDDI може означати різні речі з ідеї, що інтерфейс елемента в документі XHTML.
• І, нарешті, сенс частини WSDL 2.0 документу не визначено. Хоча елемент інтерфейсу може сформувати єдиний документ XML, він не є документом WSDL 2.0, так що його значення в значній мірі невизначене.
Програми, які вимагають такого рівня компенсації (або розкладу) у все більшій мірі заснований на RDF [*RDF*], заснований на графах мови подання знань і Web Ontology Language (OWL) [*OWL*], яка може розглядатися як розширена мова схем для RDF. По суті справи, WSDL 2.0 документ представлений в RDF може бути легко розширений довільними твердженнями RDF і WSDL 2.0 Інформація може бути легша, пов'язана з довільними іншими знаннями.

5.5.1 RDF подання WSDL 2.0

WSDL 2.0: Відображення в RDF [*WSDL 2.0 RDF картування*] описує WSDL 2.0 конструкцій може бути виражено в RDF використання класів ресурсів (описано з онтології виражається в OWL) і твердження. Як RDF представляє знання, використовуючи ресурси та відносини між ними, ми повинні перетворити WSDL 2.0 концепцію у цій моделі. Це робиться в такий спосіб.
1. По-перше, всі компоненти в WSDL 2.0 (наприклад, інтерфейси, операції, кріплення, послуги, кінцеві точки і т.д., включаючи розширення) включені до ресурсів ототожнення з відповідними URI, створені відповідно до додаванням С IRI-Документи WSDL 2.0 компонентів [*WSDL 2.0 Core*].
2. Крім того, речі, представлені як ресурси:
    a. Елементи присутні з XML Schema (або подібним чином, інші компоненти з інших систем типу)
    b. Зміст повідомлення моделями
    c. Обмін повідомленнями шаблонів (URI виявлення МЕП є URI ресурсу)
    d. Операція стилю (як URI операції стилі URI ресурсу)
3. Всі ресурси вище, відповідні типи використання RDF: тип звітності (інтерфейс буде належати до класу інтерфейсу і операція в інтерфейсі буде належати до класу InterfaceOperation, наприклад).
4. Всі відносини в WSDL 2.0 (наприклад, операції приналежності до інтерфейсу і мають ту чи іншу операцію стилю) перетворюються в заявах RDF використовуючи відповідні властивості, такі як `експлуатація` та `operationStyle`.

5.6 Нотатки про URIs

5.6.1 Простір імен в XML і схеми проведення

Це загальноприйнята помилка ототожнення цільового простору імен XML-схеми або вартості `xmlns` атрибуту в XML випадках з розміщенням відповідної схеми. Хоча простори імен URIs можуть бути можливими, щоб отримати від такої схеми розташування, це не значить, що отримана схема *єдина* схема, яка пов'язана з цим простором імен. Там може бути безліч схем, пов'язаних із певним простором імен, щоб визначити, яку з них використовувати в конкретних умовах обробки.

Специфікація WSDL 2.0 забезпечує обробку контексту тут через механізм `імпорту`, який заснований на XML Schema терміну за аналогічною концепцією.

5.6.2 Відносні URIs

У цьому документі є повні URI, вони використовуються в WSDL 2.0 і XSD прикладах. У деяких випадках, повні URI, були використані лише для ілюстрації посилань концепції. Однак, використання відносних URI, НЕ допускається, і виправдане у багатьох випадках. За інформацією з переробки відносний URI, див RFC3986.

5.6.3 Тимчасові URIs, які проводяться

В цілому, коли WSDL 2.0 документ опубліковано для використання іншими особами, він повинен містити тільки URI, яке в усьому світі унікальне. Це звичайно робиться шляхом виділення їх під ім'я домену, який перебуває під контролем емітента. Наприклад, W3C виділяє імена URI, під його ім'ям домену бази, w3.org.
Однак, іноді бажано, щоб скласти тимчасове URI для суб'єкта, для використання в ході розробки, але не роблять URI глобально унікальні для всіх часів і його "середній"(схема, документ WSDL 2.0, і т.д. .). *Верхній рівень захищений DNS іменем* [*IETF RFC 2606*] вказує деякі базові URI імена, які зарезервовані для використання цього типу поведінки. Наприклад, базовий URI "http://example.org/" може бути використаний для побудови тимчасових URI без будь-якої унікальної Асоціації особи. Це означає, що дві людини або програми, можуть вибирати одночасно використовувати тимчасові URI "http://example.org/userSchema" для двох абсолютно різних схем. До тих пір, як сфера застосування цих URI, не перетинається, то вони будуть досить унікальним. Однак, це не рекомендується, що "http://example.org/" використовуватися в якості бази.

6.Список літератури

6.1Нормативні посилання

[IETF RFC 2119]
Ключові слова для використання в RFC, для позначення рівнів вимог, C. Bradner, Автор. Internet Engineering Task Force, березень 1997 року. Доступно за адресою: http://www.ietf.org/rfc/rfc2119.txt

[IETF RFC 3023]
XML Media Types, M. Murata, C. St. Лоран, Д. Кона, авторів. Internet Engineering Task Force, січень 2001 року. Доступно за адресою: http://www.ietf.org/rfc/rfc3023.txt

[IETF RFC 3986]
Uniform Resource Identifier (URI): Generic Syntax, Т. Бернерс-Лі, Р. Філдінг, Л. Masinter, авторів. Internet Engineering Task Force, січень 2005 року. Доступно за адресою: http://www.ietf.org/rfc/rfc3986.txt

[IETF RFC 3987]
Інтернаціоналізація ідентифікаторів ресурсів (IRIs), M. Duerst, M. Suignard, авторів. Internet Engineering Task Force, січень 2005 року. Доступно за адресою: http://www.ietf.org/rfc/rfc3987.txt

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Fourth Edition), Т. Брей, J. Paoli, CM Сперберг-Маккуїн, Е. Малер, редактори. Консорціум World Wide Web, 10 лютого 1998 року, переглянутої 16 серпня 2006. Ця версія XML 1.0 Рекомендації http://www.w3.org/TR/2006/REC-xml-20060816. В останній версії XML 1.0 доступна на http://www.w3.org/TR/xml.

[XML Information Set]

XML Information Set (Second Edition), Ж. Р. Коуен і Тобіна, редактори. Консорціум World Wide Web, 24 жовтня 2001 року, до редакції 4 лютого 2004. Ця версія XML Information Set Рекомендації http://www.w3.org/TR/2004/REC-xml-infoset-20040204/. Остання версію XML Information Set доступна на http://www.w3.org/TR/xml-infoset.

[Простору імен XML]

Простори імен в XML 1.0 (Second Edition), Т. Брей, Д. Холландер, А. профана, Р. Тобіна, редакторів. Консорціум World Wide Web, 14 січня 1999 року, переглянутої 16 серпня 2006. Ця версія Простору імен в XML 1.0 Рекомендації http://www.w3.org/TR/2006/REC-xml-names-20060816/. Остання версія Простору імен в XML можна знайти на http://www.w3.org/TR/xml-names.

[XML Schema Структури]

XML Schema Part 1: Структури друге видання, Н. Томпсон, Д. Бук, М. Малоні, М. Мендельсона, редактори. Консорціум World Wide Web, 2 травня 2001 року, переглянутий 28 жовтня 2004. Ця версія XML Schema Part 1 рекомендація http://www.w3.org/TR/2004/REC-xmlschema-1-20041028. З останньою версією XML Schema Part 1 можна ознайомитися на http://www.w3.org/TR/xmlschema-1.

[XML Schema типи даних]

XML Schema Part 2: Datatypes Друге видання, з Байроном і А. Малхотра, редакторів. Консорціум World Wide Web, 2 травня 2001 року, переглянутий 28 жовтня 2004. Ця версія XML Schema Part 2 Рекомендації http://www.w3.org/TR/2004/REC-xmlschema-2-20041028. Останньою версію XML Schema Part 2 можна отримати на http://www.w3.org/TR/xmlschema-2.

[WSDL 2.0 Core]

Web Services Description Language (WSDL) 2.0 Частина 1: Основна мова, Р. Chinnici, JJ. Море, А. Райман, С. Weerawarana, редактори. Консорціум World Wide Web, 26 червня 2007 року. Ця версія "Рекомендації Web Services Description Language (WSDL) 2.0 Частина 1: Основна мова" доступна доступна на http://www.w3.org/TR/2007/REC-wsdl20-20070626. Останню версію "Web Services Description Language (WSDL) 2.0 Частина 1: Основна мова" можна отримати на http://www.w3.org/TR/wsdl20.

[WSDL 2.0 добавки]

Web Services Description Language (WSDL) 2.0 Частина 2: придатки, Р. Chinnici, H. Haas, А. Льюіс, JJ. Море, Д. Сад, С. Weerawarana, редактори. Консорціум World Wide Web, 26 червня 2007 року. Ця версія "Рекомендації Web Services Description Language (WSDL) 2.0 Частина 2: добавки" доступна на http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626. Остання версія "Web Services Description Language (WSDL) 2.0 Частина 2: добавки" доступна на http://www.w3.org/TR/wsdl20-adjuncts.

[WSDL 2.0 SOAP 1.1 обов'язкову]

Web Services Description Language (WSDL) 2.0 SOAP 1.1 Binding, А. Vedamuthu, Editor. Консорціум World Wide Web, 26 червня 2007 року. Ця версія "Web Services Description Language (WSDL) 2.0 SOAP 1.1 Binding" Примітки робочої групи можна ознайомитися на http://www.w3.org/TR/2007/NOTE-wsdl20-soap11-binding-20070626. З останньою версію "Web Services Description Language (WSDL) 2.0 SOAP 1.1 Binding" можна ознайомитися на http://www.w3.org/TR/wsdl20-soap11-binding.

[WSDL 2.0 RDF картування]

Web Services Description Language (WSDL), версія 2.0: RDF карт, J. Копецьки, Б. Parsia, редактори. Консорціум World Wide Web, 26 червня 2007 року. Ця версія "Web Services Description Language (WSDL), версія 2.0: RDF Mapping" Примітка Робочої групи можна ознайомитися на

http://www.w3.org/TR/2007/NOTE-wsdl20-rdf-20070626. З останньою версією "Web Services Description Language (WSDL), версія 2.0: RDF Mapping" можна ознайомитися на http://www.w3.org/TR/wsdl20-rdf.

[Web архітектура]
Архітектура World Wide Web, Volume One, Ян Якобс, Norman Walsh, редактори. Рекомендація W3C, 15 грудня 2004 року. Наявність на http://www.w3.org/TR/2004/REC-webarch-20041215/.

[Стара архітектура]
Web Services Architecture, Девід Бут, та ін., редактори. W3C Working Group Note, 11 лютого 2004 року. Наявність на http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[Стара Глосарія]
Web Services глосарії, Хьюго Хаас, Аллен Браун, редакторів. W3C Working Group Note, 11 лютого 2004 року. Наявність на http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.

[Опис вмісту двійкових даних в XML]
Опис вмісту двійкових даних в XML, Аніша Кармаркара Уміт Yalçınalp, редактори. W3C Робоча група Примітка 4 травня 2005 року. Наявність на http://www.w3.org/TR/xml-media-types/

6.2 Інформаційний список

[IETF RFC 2606]
Верхній рівень захищений DNS-іменами, Д. Eastlake, A. Panitz, автори. Мережева робоча група Internet Engineering Task Force, червень 1999 року. Доступно за адресою: http://www.ietf.org/rfc/rfc2606.txt.

[IETF RFC 2616]
Hypertext Transfer Protocol - HTTP/1.1, R. Fielding, J. Геттіс, J. Mogul, H. Frystyk, Л. Masinter, П. Лич, Т. Бернерс-Лі, авторів. Internet Engineering Task Force, червень 1999 року. Доступно за адресою: http://www.ietf.org/rfc/rfc2616.txt.

[IETF RFC 2818]
HTTP За TLS, E. Rescorla, Автор. Internet Engineering Task Force, May 2000. Доступно за адресою: http://www.ietf.org/rfc/rfc2818.txt.

[SOAP 1.1]
Simple Object Access Protocol (SOAP) 1,1, Д. Box, Д. Ehnebuske, Г. Kakivaya, A. Мирянин, М. Мендельсона, H. Frystyk Нільсен, C. Thatte, Д. Вінер, редактори. Консорціум World Wide Web, 8 травня 2000 року. Ця версія Simple Object Access Protocol 1,1 Примітка є http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[SOAP 1.2 Part 1: Messaging Framework]
SOAP Version 1.2 Part 1: Messaging Framework (друге видання), M. Gudgin, M. Хедлі, М. Мендельсона, JJ. Море, H. Frystyk Нільсен, редакторів. Консорціум World Wide Web, 24 червня 2003 року, переглянутої 27 квітня 2007. Ця версія "SOAP Version 1.2 Part 1: Messaging Framework" Рекомендації http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. Останню версію "SOAP Version 1.2 Part 1: Messaging Framework" можна ознайомитися на http://www.w3.org/TR/soap12-part1/.

[SOAP 1.2 Part 2: добавки]
SOAP Версія 1.2 Частина 2: добавки, M. Gudgin, M. Хедлі, М. Мендельсона, JJ. Море, H. Frystyk Нільсен, редакторів. Консорціум World Wide Web, 7 травня 2003 року, переглянутої 27 квітня 2007. Ця версія "SOAP Версія 1.2 Частина 2: добавки" Рекомендації http://www.w3.org/TR/2007/REC-soap12-part2-20070427/. Остання версія "SOAP Версія 1.2 Частина 2: добавки" доступна на http://www.w3.org/TR/soap12-part2/.

[SOAP MTOM]

SOAP оптимізації передачі повідомлень механізмів, M. Gudgin, M. Мендельсона, M. Ноттінгем, H. Ruellan, редактори. Консорціум World Wide Web, 25 січня 2005 року. Ця версія SOAP оптимізації передачі повідомлень існує механізму, який в http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/ http://www.w3.org/TR/2005/REC-soap12 -MTOM-20050125 /.

[WSD вимоги]

Web Services Description вимоги, J. Schlimmer, Editor. Консорціум World Wide Web, 17 жовтня 2002 року. Ця версія документу Web Services Description вимог http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028. З останньою версією Web Services Description вимог можна ознайомитися на http://www.w3.org/TR/ws-desc-reqs.

[WS-Addressing]

Web Services Addressing 1.0 - Core, Мартін Gudgin, Марк Хедлі, редактор. Консорціум World Wide Web, 17 серпня 2005 року. Ця версія Web Services Addressing 1.0 - Core документ доступний http://www.w3.org/TR/ws-addr-core/. З останньою версією Web Services Description вимог можна ознайомитися на http://www.w3.org/TR/ws-addr-core/.

[XPointer рамок]

Рамкова XPointer, Пол Гросс, Єва Малер, Jonathan Marsh, Norman Walsh, редактори. Консорціум World Wide Web, 25 березня 2003 року. Ця версія XPointer рамки, запропоновані рекомендації http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ Остання версія XPointer рамки наявна на http://www.w3.org/ TR / xptr рамки /.

[W3C TAG Висновок: Використання HTTP GET]

URI, адресацію і використання HTTP GET та POST, Ян Якобс, редактор. Консорціум World Wide Web, 21 березня 2004 року. Наявність на http://www.w3.org/2001/tag/doc/whenToUseGet

[W3C TAG Висновок: Розширення і версіями Мови Частина 1]

Розширення і версіями Мови частина 1 David Orchard, редактор. Консорціум World Wide Web, 26 березня 2006 року. Наявність на http://www.w3.org/2001/tag/doc/versioning

[WebArch: Extensible Мови]

Веб-архітектура: Extensible Мови, Тім Бернерс-Лі, Ден Конноллі, авторів. W3C Note 10 лютого 1998. Наявність на http://www.w3.org/TR/NOTE-webarch-extlang

[XML Schema: Версія приклади використання]

XML Schema версіями прецедентів, Hoylen Сью. W3C XML Schema робочої групи проекту, 31 січня 2006 року. Наявність на http://www.w3.org/XML/2005/xsd-versioning-use-cases/

[Посібник з версіями XML Мови допомогою XML Schema 1.1]

Посібник з версіями XML Мови допомогою XML Schema 1.1, David Orchard. W3C XML Schema робочої групи проекту, 28 вересня 2006 року. Наявність на http://www.w3.org/TR/xmlschema-guide2versioning

[XML Schema 1.1]

XML Schema 1.1 Частина 1: Структури, H. Томпсон, CM Сперберг-Маккуїн, Shudi (Sandy) Гао, M. Мендельсона, David Beech, Murray Maloney, редактори. Консорціум World Wide Web, 31 серпня 2006 року. Це робочий проект XML Schema 1.1 Частина 1 http://www.w3.org/TR/2006/WD-xmlschema11-1-20060831/. Остання версія XML Schema 1.1 частини 1 доступна на http://www.w3.org/TR/xmlschema11-1/.

[SW VocabManagementNote]

Словник управління, Томас Бейкер, та ін. Semantic Web Best Practices і розгортання робочої групи Відзначимо, 8 лютого 2005. Наявність на http://esw.w3.org/topic/VocabManagementNote

[RELAX NG]

RELAX NG специфікації, Джеймс Кларк, Murata Макото, редакторів. Комітет специфікація OASIS, 3 грудня 2001 року. Наявність на http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[JAX RPC 1.1]

Java (TM) API на основі XML Remote Procedure Call (JAX-RPC) специфікацію, версії 1.1, Роберто Chinnici та ін. 14 жовтня 2003. Наявність на http://java.sun.com/xml/downloads/jaxrpc.html

[Відпочинку]

Образотворчі State Transfer (REST), Рой Томас Філдінг, автор. 2000. Наявність на http://www.ics.uci.edu/ ~ Fielding / Pubs / дисертації / Top.htm

[RDF]

Resource Description Framework (RDF): Поняття і абстрактного синтаксису, Грем Klyne, Джеремі J. Carroll, редакторів. Рекомендація W3C, 10 лютого 2004 року. Наявність на http://www.w3.org/TR/rdf-concepts/

[OWL]

OWL Web Ontology Language номер, Майк Дін, Гус Шрайбер, редакторів. Рекомендація W3C від 10 лютого 2004. Наявність на http://www.w3.org/TR/owl-ref/

[Альтернативна схема мови підтримки]

Обговорення альтернативних схем мови і типу системної підтримки в WSDL, А. Льюіс, Б. Parsia, редактори.

Подяка

Даний документ являє собою роботу W3C Web Опис послуг Робочої групи.
Члени Робочої групи (на момент написання, а в алфавітному порядку): Чарльтон Баррето (Adobe Systems, Inc), Аллен Брукс (Rogue Wave Softwave), Dave Chappell (Sonic Software), Хелен Чен (Agfa-Геварт H.B. "), Роберто Chinnici (Sun Microsystems), Кендалл Кларк (Університет Меріленда), Глен Денielс (Sonic Software), Поль Дауні (Британські телекомунікації), Youenn Fablet (Canon), Рам Jeyaraman (Microsoft), Том Jordahl (Adobe Systems), Аніша Кармаркара (Oracle Corporation), Яцек Копецьки (DERI Інсбрук на Леопольд-Франценс університет Інсбрука, Австрія), Амелія Льюіс (TIBCO Software, Inc), Філіп Ле Hegaret (W3C), Майкл Liddy (Education.au TOB) Кевін Canyang Лю (SAP AG), Джонатан Марш (WSO2), Моніка Мартін (Sun Microsystems), Жозефіна Мікаллеф (SAIC - Telcordia технологій), Джефф Mischkinsky (Oracle Corporation), Дейл Моберг (Cyclone Commerce), Жан-Жак Моро (Canon), David Orchard (BEA Systems, Inc), Жильбера Pilz (BEA Systems, Inc), Тоні Роджерс (Computer Associates), Артур Райман (IBM), Аді Сакала (IONA Technologies), Майкл Шеперд (Xerox), Асир Vedamuthu (Microsoft Corporation), Санджіву Weerawarana (WSO2), Юміт Yalçınalp (SAP AG), Пітер Zehler (Xerox).
Попередній входили: Еран Chinthaka (WSO2), Марк Ноттінгем (BEA Systems, Inc), Хьюго Хаас (W3C), Вівек Панді (Sun Microsystems), Bijan Parsia (Університет Меріленда), Лілі Ліу (WebMethods, Inc), Дона Райта (Lexmark), Джойс Янг (Oracle Corporation), Даніель Schutzer (Citigroup), Дейв Індивідуальний (Citigroup), Стефано Pogliani (Sun Microsystems), Вільям Stumbo (Xerox), Стівен Білий (SeeBeyond), Барбара Zengler (DaimlerChrysler і науково-дослідного Технологія "), Тім Finin (Університет Меріленда), Лоран де Teneuille (L'Echangeur), Йохан Pauhlsson (L'Echangeur), Марк Джонс (AT & T), Стів Лінд (AT & T), Сандра Swearingen (міністерство оборони США, US Air сил), Філіп Ле Hégaret (W3C), Джим Хендлер (Університет Меріленда), Дітмар Gaertner (Software AG), Майкл чемпіон (Software

AG), "Дон Маллен (TIBCO Software, Inc), Стівен Грем (Global Grid Forum), Стів Tuecke (Global Grid Forum), Майкл Махан (Nokia), Брайан Томпсон (Хікс & Associates), Інго Мельцер (DaimlerChrysler досліджень і технологій), Sandeep Кумар (Cisco Systems), Алан Девіс (SeeBeyond), Яцек Копецьки (Systinet), Mike Баллантайн (Electronic Data Systems), Майк Davoren (WW Grainger), Ден Кулп (IONA Technologies), Майк Макью (WW Grainger), Майкл Mealling (Verisign), Вакар Садок (Electronic Data Systems), Ярон Голанд (BEA Systems, Inc .) Уміт Yalçınalp (Oracle Corporation), Пітер Madziak (Agfa-Gevaert H.B.), Джеффрі Schlimmer (Microsoft Corporation), Хао Хе (Thomson Corporation), Ерік Акерман (Lexmark), Джері Thrasher (Lexmark), Прасад Yendluri (WebMethods , Inc), Вільям Vambenepe (Hewlett-Packard Development Company), Девід Бут (W3C), Санджіву Weerawarana (IBM), Асир Vedamuthu (WebMethods, Inc), Ігор Sedukhin (Computer Associates), Мартін Gudgin (Microsoft Corporation), Ребекка Bergersen (IONA Technologies), Уго Корда (SeeBeyond).
Людям, які внесли свій внесок в обговорення www-ws-desc@w3.org також подяка.