پشتیبانی تراکنش‌ها در ترکیب سرویس‌های مبتنی‌بر وب

فاطمه چیت‌فروش[۱]، مریم یزدان‌دوست[۲]، حسن ابوالحسنی [۳]
[۱] chitforoush@ce.sharif.edu.
[۲] yazdandoost@ce.sharif.edu
[۳] abolhassani@ce.sharif.edu

۱- مقدمه

از مطرح‌ترین فناوری‌های ارائه‌شده در سطح وب، سرویس‌های وب هستند. سرویس‌های وب برنامه‌های کاربردی هستند که توسط انسان و سایر برنامه‌ها از طریق وب، و به شکل مستقل از بستر و زبان برنامه‌نویسی قابل دسترس هستند. از طرف دیگر وب معنایی در سال ۲۰۰۱ با هدف تولید مکانیزمی برای نمایش اطلاعات به‌شکل قابل درک برای عوامل غیرانسانی مطرح شده‌است. از ترکیب دو تکنولوژی سرویس‌های وب و وب‌معنایی، سرویس‌های وب معنایی مطرح شده‌اند. از طرف دیگر با پیچیده‌شدن نیازمندی‌ها، ترکیب سرویس‌های وب متعدد جهت ایجاد سرویسی با ارزش افزوده مطرح شده‌است. در این فرآیند که با درخواست کاربر برای یک سرویس مرکب آغاز شده، پس از بررسی درخواست وی و انتخاب سرویس‌های مناسب، توصیفی از سرویس مرکب موردنظر توسط یک زبان مناسب ارائه می‌شود. موتور اجرای سرویس مرکب، براساس این تعریف، سرویس مرکب را اجرا می‌کند. درحین اجرای یک سرویس مرکب ممکن است، هر یک از سرویس‌های تشکیل‌دهنده‌ی سرویس مرکب دچار مشکلاتی شوند، که ادامه‌ی اجرای سرویس را غیرممکن سازد. موتور اجرا باید بتواند اجرای سرویس مرکب به شکل یک تراکنش را تضمین کند.

تراکنش به شکل کلاسیک به مجموعه‌ی اعمالی اطلاق می‌گردد که وضعیت را به گونه‌ای تغییر می‌دهند که چهار ویژگی را داراست: اتمیک بودن[۱]، سازگاری[۲]، جدا بودن[۳] و ماندگاری[۴] [1]. مدل تراکنشی ACID برای محیط سرویس‌های وب با پیوستگی ضعیف[۵] مناسب نیست و بسیار سخت‌گیرانه به نظر می‌رسد. در نتیجه مدل‌هایی برای رفتار تراکنشی سرویس‌های مرکب ارائه شده‌است.

در این مقاله مدلی برای رفتار تراکنشی در سرویس‌های مرکب ارائه شده‌است. با توجه به ناهمگونی در محیط سرویس‌های وب، رفتارهای تراکنشی مختلفی برای سرویس‌ها با جزئیات کامل تعریف شده‌است. به‌علاوه مدل ارائه‌شده، به شکل گسترشی بر هست-شناسی OWL-S مدل شده‌است، که با توجه به وجود ابزارهای متعددی که OWL-S را پشتیبانی می‌کنند، به‌کارگیری مدل در برنامه‌های کاربردی ممکن می‌شود.

همچنین در این مقاله الگوریتمی ارائه شده‌است که بر اساس مدل رفتار تراکنشی ارائه‌شده، در صورت بروز خطاهای زمان اجرا برای هریک از سرویس‌ها، اثرات سرویس‌هایی که در قالب سرویس مرکب تا قبل از بروز خطا اجرا شده‌اند را تا حد ممکن خنثا می‌کند.

در ادامه، در بخش دوم مروری بر مدل‌های تراکنشی مطرح خواهیم‌داشت. در بخش سوم، مدل رفتار تراکنشی ارائه‌شده در این مقاله را تعریف می‌کنیم و در بخش چهارم گسترش پیشنهادشده بر هست‌شناسی OWL-S بر اساس مدل ارائه‌شده را بررسی می‌کنیم. در بخش پنجم، الگوریتم ارائه‌شده برای بازیابی از خطاهای زمان اجرا را خواهیم دید و در پایان بخش ششم به نتیجه‌گیری اختصاص دارد.

۲- بررسی مدل‌های تراکنشی ارائه‌شده

تراکنش به شکل کلاسیک به مجموعه‌ی اعمالی اطلاق می‌گردد که وضعیت را به گونه‌ای تغییر می‌دهند که چهار ویژگی را داراست: اتمیک بودن، سازگاری، ماندگاری و جدابودن. اتمیک بودن به معنی اجرای «همه یا هیچ‌کدام» از عملیات‌هاست، سازگاری تغییر وضعیت صحیح را تضمین می‌کند، جدا بودن به معنی عدم تداخل بین تراکنش‌هایی است که هم‌زمان اجرا می‌شوند و ماندگاری نیز به

آن معنی است که یک تراکنش بعد از اتمام، قابل فسخ‌کردن نیست. برهمین اساس تراکنش‌های کلاسیک به تراکنش‌های ACID معروفند.

مدیریت تراکنش‌ها در محیط سرویس‌های وب مشابه مدیریت تراکنش‌ها در سایر محیط‌های توزیع شده است، با این تفاوت که در چنین محیط‌هایی پیوستگی بسیار ضعیف است و سرویس‌ها نیز قابل اطمینان نیستند. به همین دلیل و به دلیل ناهم‌گونی سرویس‌ها در محیط وب برای چنین محیط‌هایی تراکنش‌های ACID مناسب به‌نظر نمی‌رسند. از آن‌جایی‌که تراکنش‌ها در محیط سرویس‌های وب معمولاً بسیار طولانی‌مدت هستند، قفل‌کردن منابع جهت اجرای یک سرویس نیز ممکن نیست که این نیز با به‌کارگیری تراکنش-های ACID قطعاً در تناقض است، ضمن این‌که سیستم‌ها معمولاً اجازه‌ی قفل‌کردن منابع توسط سرویس‌های خودمختار وب را نمی‌دهند.

با توجه به محدودیت‌های ذکرشده در محیط سرویس‌های وب، نوع ضعیف‌تری از تراکنش‌ها با عنوان "تراکنش‌های طولانی‌مدت"[6] مطرح شدند که در آن‌ها نیاز به ضعیف‌کردن مفهوم بازگشت به عقب[7]، باعث مطرح‌شدن مفهوم خنثاکردن شد. خنثاکننده‌ی تراکنش T خود یک تراکنش C است، که بعد از پایان تراکنش T و جهت خنثاکردن تغییرات آن، در صورت بروز خطا اجرا می‌شود. به همین دلیل نیازی به انجام تغییرات موقتی و قفل‌کردن منابع توسط تراکنش T نیست. برخلاف بازگشت به عقب که جزئی از یک تراکنش به شمار می‌رود، خنثاکننده خود تراکنش مستقلی است که بعد از پایان تراکنش T و خارج از حوزه‌ی این تراکنش اجرا می‌شود. مقداری که تراکنش خنثاکننده می‌تواند اثرات تراکنش T را خنثا کند کاملاً به حوزه‌ی خاص تراکنشT وابسته است، در برخی از سناریوها ممکن است این تغییرات کاملاً قابل خنثاکردن باشند و در برخی موارد دیگر مثل پاک‌کردن اطلاعات و یا ارسال نامه‌ی الکترونیکی اثرات قابل‌جبران نباشند.

در ادامه به بررسی مهم‌ترین مدل‌های تراکنشی ارائه شده می‌پردازیم.

۲-۱- WS-Transaction

چارچوب WS-Transaction [2] به دو قسمت اصلی تقسیم می‌شود که هریک نوعی رفتار تراکنشی را مدل می‌کنند WS-AtomicTransaction [3] تراکنش‌های ACID را مدل می‌کند و WS-BusinessActivity [4] تراکنش‌های طولانی‌مدت را مدل می‌کند. این دو چارچوب هر دو از WS-Coordination [5] استفاده می‌کنند، که ایجاد یک زمینه‌ی[8] مشترک در یک محیط توزیع شده را پیاده‌سازی می‌کند.

یک تراکنش اتمیک[9] یا AT برای مدل‌کردن فعالیت‌هایی که در زمانی کوتاه و در دامنه‌هایی محدود با مراقبت‌های بالا اجرا می‌شوند به کار می‌رود. این تراکنش‌ها برای کاربردهایی با فعل و انفعالات[10] کم و شامل سرویس‌هایی که بر روی تبعیت از ویژگی‌های ACID موافقت دارند، قابل پیاده‌سازی است. هر یک از سرویس‌های شرکت‌کننده در تراکنش، رکوردهای مربوطه در پایگاه داده را در طول اجرای تراکنش منتقل می‌کنند. هماهنگ‌کننده تراکنش پروتکل خاتمه دومرحله‌ای[11] را برای تمامی سرویس‌های شرکت‌کننده پیاده-سازی می‌کند. همان‌طور که گفتیم این روش شامل معایب قفل‌کردن منابع می‌شود.

WS-AtomicTransaction سه نوع پروتکل هماهنگی را تعریف می‌کند: «Completion» برای آغاز و یا خاتمه‌دادن توسط یک سرویس استفاده می‌شود، «خاتمه‌ی دو مرحله‌ای فرار[12]» برای داده‌های فرار مثل داده‌های حافظه Cache و «خاتمه‌ی دو مرحله‌ای بادوام[13]» برای پیاده‌سازی پروتکل خاتمه‌ی دو مرحله‌ای استاندارد.

WS-BusinessActivity فعالیت تجاری[14] را به عنوان ترکیبی از تراکنش‌های اتمیک تعریف می‌کند. بازگشت به عقب برای فعالیت-های تجاری را با استفاده از مفهوم سرویس‌های خنثاکننده پیاده‌سازی می‌کند. با استفاده از فعالیت‌های تجاری می‌توان تراکنش‌هایی سلسله‌مراتبی تعریف کرد که نیازی به قفل‌کردن تمامی رکوردهای پایگاه داده ندارند. WS-BusinessActivity دونوع پروتکل ارتباطی را پیاده‌سازی می‌کند:Business-AgreementWithParticipantCompletion که در آن سرویسی که تراکنش را آغاز کرده است، پایان تراکنش را تعیین می‌کند و BusinessAgreement-WithCoordinatorCompletion که درآن پایان تراکنش توسط هماهنگ‌کننده‌ی بیرونی مدیریت می‌شود.

www.SID.ir

۲-۲- WS-TXM

چارچوب WS-CAF [6] که توسط شرکت Sun ارائه شده‌است شامل سه بخش است، که با معماری لایه‌ای بر روی یکدیگر قرار گرفته‌اند. بخش WS-TXM بالاترین لایه در این چارچوب است که مسئولیت مدیریت تراکنش‌ها را بر عهده دارد. این چارچوب سه مدل مختلف برای چشتیبانی رفتارهای تراکنشی ارائه کرده است: ۱) **TX-ACID**: این مدل جهت ایجاد قابلیت همکاری[۱۵] با سیستم-های پردازش تراکنش‌ها، که در برنامه‌های مؤسسات بزرگ از قبل باقی مانده‌اند، از طریق سرویس‌های وب طراحی شده‌است. همان‌طور که قبلاً هم گفتیم، این مدل تراکنش‌ها قطعاً برای تمامی سناریوها مناسب نیستند. اما هنوز هم کاربردهای خاصی دارند، به‌ویژه برای تراکنش‌های اقتصادی- مالی که شامل تعاملاتی با بار مالی زیاد هستند، ۲) **TX-LRA**: برای به‌کارگیری در تعاملات تجاری طولانی-مدت طراحی شده‌است و در آن هر فعالیت[۱۶] تعاملات تجاری را مدل می‌کند. در این مدل تراکنش‌ها قابل خنثاکردن هستند، اما اینکه هر سرویس وب چگونه و تا چه حد خنثاکردن اثراتش را پشتیبانی و پیاده‌سازی می‌کند، در حوزه‌ی وظایف تأمین‌کننده‌ی سرویس قرار دارد و مدل LRA تنها شرایط و زمان اجرای سرویس‌های خنثاکننده را براساس تعریف سرویس مرکب مشخص می‌کند، و ۳) **TX-BP**: در این مدل تمامی شرکت‌کنندگان در یک فرآیند تجاری در «حوزه‌های تجاری» قرار دارند. تراکنش‌های فرآیند تجاری مسئول مدیریت تعاملات بین این حوزه‌ها هستند. هر فرآیند تجاری به «وظایف تجاری» تقسیم می‌شود که هر یک درحوزه‌ی تجاری خاصی اجرا می‌شود. هر «وظیفه» ممکن است توسط سرویس‌های متعددی انجام شود، به‌علاوه هر وظیفه باید قابل خنثاکردن باشد. در این مدل «کنترل‌کننده» از حوزه‌های تجاری می‌خواهد به شکل دوره‌ای وضعیت خود را ثبت کنند تا با استفاده از آن درصورت بروز خطا بتوان ادامه عملیات را مجدداً از نقطه‌ای که وضعیت ثبت شده ادامه داد.

۲-۳- دیدگاه وضعیت‌های پایانی قابل قبول

در دیدگاه وضعیت‌های پایانی قابل قبول[۱۷] [7]، از ویژگی وضعیت‌های پایانی قابل قبول برای بررسی صحت وضعیت پایانی از لحاظ نقض شرایط اتمیک‌بودن استفاده می‌شود. انواع رفتارهای تراکنشی که در این دیدگاه درنظر گرفته شده‌اند عبارتند از: قابل تلاش مجدد[۱۸]، قابل خنثاکردن[۱۹] و محور اصلی[۲۰]. سرویسی قابل‌تلاش‌مجدد است که حداکثر بعد از تعداد دفعات مشخصی اجرا، درست اجرا شود. سرویس قابل خنثاکردن سرویسی است که اثرات آن از لحاظ معنایی قابل خنثاکردن است و سرویس محور اصلی بعد از خاتمه به هیچ‌عنوان قابل خنثاکردن نیست.

در این رویکرد برای بیان رفتار داخلی سرویس، از نمودار حالت[۲۱] استفاده می‌شود. نمودار حالت هر سرویس شامل مجموعه‌ی مشخصی از حالت‌ها و تعدادی گذار بین حالات می‌شود. رفتار تراکنشی خاصی که در یک کاربرد خاص مورد نیاز است، با گسترش این نمودار حالت قابل مدل‌کردن است. برای ترکیب و همزمانی سرویس‌های جزء در یک سرویس مرکب و هم‌چنین برای تعریف مکانیزم برخورد با خطاهای زمان اجرا از همین نمودارها استفاده می‌شود. در این مدل اجرای سرویس مرکب تنها زمانی‌که به یک حالت پایانی قابل قبول ختم شود صحیح درنظر گرفته می‌شود. یک سرویس مرکب درصورتی به یک حالت پایانی قابل‌قبول ختم می‌شود که یا موفقیت آمیز خاتمه یابد، و یا دچار خطا شده ولی تمامی تأثیرات نامطلوب اجرای ناقص آن، خنثا شود.

۲-۴- BTP

پروتکل BTP[۲۲] [8] بین المان‌های کاربردی (یا سرویس‌های وب) و المان‌های BTP که مسئول هماهنگ‌کردن تراکش‌ها هستند تمایز قائل می‌شود. المان‌های BTP ممکن است نودی باشند که یک تراکنش را هماهنگ می‌کنند و یا نودی باشند که خود در تراکنشی که توسط نودی دیگر هماهنگ می‌شود شرکت می‌کنند. البته این المان‌ها می‌توانند هم‌زمان هردو نقش را ایفا کنند و در نتیجه ساختارهای درختی به‌وجود آورند.

BTP نیز دو نوع تراکنش را تعریف می‌کند: تراکنش اتمیک که در آن تمامی المان‌های شرکت‌کننده در تراکنش باید نهایتاً به یک تصمیم مشترک در مورد پایان تراکنش برسند و تراکنش‌های به‌هم‌پیوسته[۲۳] که اجازه می‌دهند المان‌های شرکت‌کننده تصمیم‌های متفاوتی در مورد پایان تراکنش بگیرند. این تراکنش‌ها برای پیاده‌سازی حالاتی که یک سرویس توسط تأمین‌کنندگان متعددی ارائه می‌شود مناسب است، به این ترتیب که در زمان اجرا نهایتاً یکی از سرویس‌های ارائه شده‌ی مشابه اجرا می‌شود. این پروتکل نیز برای

مدیریت خطا از ایده‌ی خنثاکردن استفاده می‌کند، به‌علاوه در حالاتی که به هر دلیلی عملیات بازگشت به عقب ممکن نباشد، مداخله-ی دستی کاربر را ضروری می‌داند.

۳- معرفی مدل رفتار تراکنشی

در این بخش مدل رفتار تراکنشی ارائه شده در این مقاله را مطرح می‌کنیم. به جز مجموعه‌ی مدل‌هایی که در بخش قبل بررسی کردیم، تعداد زیادی مدل دیگر نیز بررسی شده‌اند، که متأسفانه به علت محدودیت، در اینجا از ذکر آن‌ها خودداری می‌کنیم. اکثر این مدل‌ها اشتراکات زیادی با مدل‌های مطرح شده دارند و حاوی نکات جدیدی نیستند [9، 10، 11، 12، 13، 14 و 15].

با توجه به محیط ناهمگن سرویس‌های وب، قطعاً با رفتارهای تراکنشی متفاوتی روبه‌رو هستیم، که باید در مدلی که ارائه می‌کنیم در نظر گرفته شوند. به‌علاوه پارامترهای مختلفی برای سرویس‌ها مطرح هستند که برای دقیق‌تر شدن عکس‌العمل الگوریتم بازیابی از خطا، این جزئیات نیز مدل شده‌اند و در الگوریتم در نظر گرفته می‌شوند. به همین دلیل این مدل از تمامی مدل‌های ارائه شده‌ی دیگر دقیق‌تر است و الگوریتمی که بر اساس این مدل نیز ارائه می‌شود، بسیار دقیق عمل می‌کند. به‌علاوه در صورت نیاز به رفتار تراکنشی خاص، که برای کاربردی خاص مناسب‌تر به نظر می‌رسد، می‌توان به راحتی این مدل را گسترش داد.

مدل رفتار تراکنشی که ارائه می‌کنیم، از جمله مدل‌های باز تو در تو است [16]، به عبارت دیگر در این مدل ویژگی جدا بودن از ویژگی‌های تراکنش‌های ACID را در نظر نمی‌گیریم، تراکنش‌های لایه‌های پایین‌تر قبل از خاتمه‌ی تراکنش سطح بالاتر منابع در دسترس خود را آزاد می‌کنند و تغییرات دائمی در سیستم ایجاد می‌کنند. این مدل بر اساس مفهوم خنثاکردن بنا نهاده شده‌است، یعنی در صورت بروز خطا در هر یک از تراکنش‌ها، اثرات سرویس‌هایی که اجرا شده‌اند، با اجرای سرویس‌های خنثاکننده‌ی آن‌ها خنثا می‌شود.

در این قسمت به مشخصه‌هایی که برای هر سرویس در این مدل در نظر گرفتیم می‌پردازیم:

- **مشخصه‌ی اتمیک بودن**[۲۴]: این مشخصه نشان می‌دهد که آیا اجرای این سرویس باید به شکل اتمیک باشد یا نه. در واقع این مشخصه برای یک سرویس مرکب مشخص می‌کند که آیا اجرای اتمیک سرویس مرکب لازم است یا نه. این مشخصه می‌تواند مقدار اتمیک یا غیر اتمیک (atomic/non-atomic) داشته باشد.
- **مشخصه‌ی قابلیت فراخوانی مجدد**[۲۵]: برخی سرویس‌ها به گونه‌ای طراحی شده‌اند که تأمین‌کننده‌ی سرویس تضمین می‌کند پس از تعداد متناهی و مشخص فراخوانی سرویس، نهایتاً باموفقیت اجرا می‌شود. در واقع تأمین‌کننده‌ی سرویس می‌داند که حتی اگر یک‌بار اجرای سرویس با خطا مواجه شد، در اجراهای بعدی، سرویس می‌تواند از راهی دیگر عملیات درخواستی را با موفقیت خاتمه دهد. برای این‌گونه سرویس‌ها مشخصه‌ی قابلیت تلاش مجدد مقدار صحیح دارد.

  از جمله اطلاعاتی که در مورد یک سرویس با قابلیت تلاش مجدد باید مدل شود، تعداد دفعاتی است که می‌توان برای اجرای صحیح سرویس تلاش کرد. این عدد متناهی توسط تأمین‌کننده‌ی سرویس تعیین می‌شود. به‌علاوه معمولاً سرویس‌ها جهت تلاش مجدد برای اجرا، محدودیت زمانی دارند، مثلاً اجرای مجدد آن‌ها تا زمانی خاص یا در یک بازه‌ی زمانی محدود، بعد از اولین اجرای سرویس، قابل انجام است.
- **مشخصه‌ی طریقه‌ی بازیابی**[۲۶]: این مشخصه وضعیت سرویس از لحاظ نحوه‌ی برخورد با خطا را مشخص می‌کند. مقادیر ممکن برای این مشخصه عبارتنداز:

۱. **بی‌تأثیر**[۲۷]: سرویس‌هایی که جهان را به گونه‌ای تغییر نمی‌دهند که در صورت بروز خطا به هرگونه عملیات بازگشت به عقب یا خنثاکردن نیاز باشد. برای مثال می‌توان به سرویس‌هایی اشاره کرد که اخبار را از منابع مختلف جمع‌آوری می‌کنند و یا قیمت ارز در بازار را بررسی می‌کنند. بدیهی است که این‌گونه سرویس‌ها نیاز به هیچ مکانیزم جبرانی در صورت بروز خطا ندارند.

۲. **قابل خنثاکردن**[۲۸]: سرویس‌هایی قابل خنثاکردن هستند که اثراتشان در جهان قابل خنثاکردن است، بدیهی است که برای سرویس‌های قابل خنثاکردن، سرویس خنثاکننده نیز باید از طرف تأمین‌کننده معرفی شود.

۳. **قطعی**[۲۹]: این‌گونه سرویس‌ها نه قابل خنثاکردنند و نه بی‌تأثیر. تأثیرات این سرویس‌ها در جهان واقعی به‌هیچ‌وجه قابل جبران نیست و به محض خاتمه‌ی چنین سرویسی، تأثیری در جهان می‌گذارد که کاملاً ماندگار و قطعی است.

با توجه به تعریف طریقه‌های بازیابی مشخص است که ترکیب‌های طریقه‌های بازیابی در یک سرویس ممکن نیست، یعنی مثلاً یک سرویس نمی‌تواند در عین اینکه قابل خنثا کردن است، بی‌تأثیر نیز باشد.

طریقه‌ی بازیابی یک سرویس مرکب نیز بر اساس طریقه‌ی بازیابی سرویس‌های تشکیل‌دهنده‌ی سرویس مرکب تعیین می‌شود. تشخیص طریقه‌ی بازیابی یک سرویس مرکب که شامل انواع مختلف سرویس‌ها با طریقه‌های بازیابی مختلف است، براساس جدول 1 انجام می‌شود. همان‌طور که می‌بینید در این جدول برای هر نوع طریقه‌ی بازیابی یک ستون در نظر گرفته شده است. ستون سمت راست طریقه‌ی بازیابی سرویس مرکبی که شامل سرویس‌هایی با طریقه‌های بازیابی شامل ستون‌های سمت چپ است را تعیین می‌کند. برای مثال ردیف اول بیان می‌کند که سرویس مرکبی که شامل سرویس‌هایی با تمامی انواع طریقه‌هایی بازیابی باشد قطعی است، چون وجود یک سرویس قطعی در یک سرویس مرکب کافی است که عملاً اثرات آن سرویس کاملاً قابل جبران‌کردن نباشد.

همان‌طور که در جدول مشخص است، برای برخی از خانه‌ها، طریقه‌ی بازیابی مشخص نیست. در واقع طریقه‌ی بازیابی این‌گونه سرویس‌ها تنها بر اساس نوع طریقه‌های بازیابی سرویس‌های تشکیل‌دهنده‌ی سرویس مرکب قابل تشخیص نیست و به پارامترهای دیگری مثل ترتیب اجرای سرویسها در سرویس مرکب وابسته است. طریقه‌ی بازیابی این سرویسها ممکن است قطعی یا قابل خنثا-کردن باشد. جزئیات مربوط به طریقه‌ی بازیابی این سرویس‌ها در بخش ۳-۲ بررسی می‌شود.

جدول ۱ – طریقه‌ی بازیابی سرویس مرکب

Affectless	Compensable	Definite	Composite
√	√	√	?
√	√	×	Compensable
√	×	√	?
√	×	×	Affectless
×	√	√	?
×	√	×	Compensable
×	×	√	?

در بخش بعدی جزئیات مربوط به طریقه‌ی بازیابی قابل خنثاکردن را بررسی می‌کنیم.

۳-۱- طریقه‌ی بازیابی قابل خنثاکردن

بر اساس مدل تراکنشی ارائه شده، عملیات خنثاکردن توسط سرویس دیگری که سرویس خنثاکننده نام دارد و توسط تأمین‌کننده‌ی سرویس معرفی می‌شود، انجام می‌گیرد. سرویس خنثاکننده اطلاعاتی را به عنوان ورودی می‌گیرد. این ورودی‌ها ممکن است جزء ورودی‌های سرویس اولیه باشند، اما ممکن است که ورودی‌هایی هم داشته باشد که جزء ورودی‌های سرویس اولیه نباشند. برا ی مثال سرویس reserveSeat() را با ورودی‌های مشتری و اطلاعات پرواز در نظر بگیرد. سرویس خنثاکننده‌ی این سرویس (cancelSeat())، به عنوان ورودی شماره صندلی را نیاز دارد، که جزء ورودی‌های سرویس اولیه نیست، بلکه خروجی سرویس reserveSeat() است. در هر حال ورودی‌های سرویس خنثاکننده، زیرمجموعه‌ای از ورودی‌ها و خروجی‌های سرویس اولیه هستند.

مکانیزم خنثاکردن برای یک سرویس ممکن است مشروط باشد («قابل خنثاکردن مشروط»)، یعنی سرویس تنها در صورت برقراربودن شرایطی، قابل خنثاکردن است. برای مثال معمولاً باز پس‌گرفتن کالای فروخته‌شده مشروط به بازگرداندن کالای خریداری-شده است. یعنی بازگرداندن پول به مشتری مشروط به بازگرداندن کالای خریداری شده به فروشنده است (مثلاً اگر سرویس اصلی

purchaseBook() باشد، خنثاکننده‌ی این سرویس مشروط به اجرای موفقیت آمیز سرویس returnBook() است). این شرایط نیز باید به نوعی همراه با سرویس خنثاکننده تعریف شوند.

معمولاً اجرای سرویس خنثاکننده برای تأمین‌کننده سرویس هزینه‌های زیادی از نظر مالی، زمانی و استفاده از منابع دارد. طبیعی است که تأمین‌کننده بخشی از این هزینه‌ها را به عنوان هزینه‌ی لغو در ازای اجرای سرویس خنثاکننده از مشتری دریافت کند. در صورتی که مشتری از لحاظ هزینه محدودیت‌هایی داشته باشد و برای مثال چند سرویس خنثاکننده‌ی مختلف با هزینه‌های مختلف برای یک سرویس وجود داشته باشد، الگوریتم بازیابی از خطا باید هزینه‌ی لغو را نیز مد نظر قراردهد.

معمولاً تأمین‌کنندگان خنثاکردن یک سرویس را در یک بازه‌ی زمانی محدود مجاز می‌دانند، برای مثال بازگرداندن کالای فروخته شده تنها تا مدت زمانی مشخص بعد از فروش کالا ممکن است. این زمان نیز باید همراه سرویس خنثاکننده مشخص شود، و در الگوریتم بازیابی از خطا نیز مد نظر قرار گیرد.

با دقت بیش‌تر می‌توان طریقه بازیابی قابل خنثاکردن را در دو گروه دسته بندی کرد:

۱. **قابل خنثاشدن کامل**[30]: این نوع طریقه‌ی بازیابی مربوط به سرویس‌هایی است که سرویس خنثاکننده‌ی آن‌ها تمامی آثار سرویس را می‌تواند خنثاکند، به شکلی که گویی از ابتدا این سرویس اجرا نشده است.

۲. **قابل خنثاشدن ناقص**[31]: گاهی سرویس خنثاکننده تنها بخشی از آثار اجرای سرویس را از بین می‌برد. در این حالت مقدار نسبی قابلیت سرویس خنثاکننده در از بین‌بردن آثار سرویس نیز باید مد نظر قرارگیرد و به همراه سرویس خنثاکننده مدل شود. این مقدار به شکل درصد خنثاشدن سرویس مدل می‌شود.

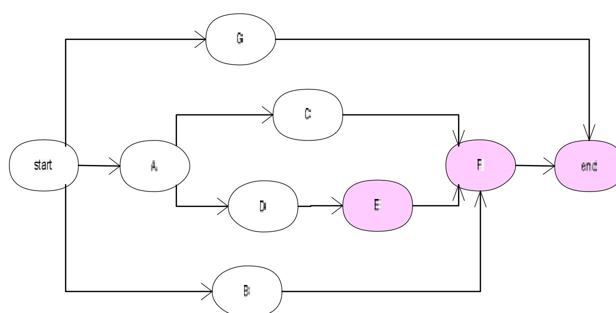
۳-۲- تعریف سرویس مرکب قابل خنثاشدن

در این بخش رفتار تراکنشی قابل خنثاشدن را برای سرویس مرکبی که شامل سرویس‌هایی با طریقه‌های بازیابی مختلف است، تعریف می‌کنیم. واضح است که پشتیبانی رفتار تراکنشی برای یک سرویس مرکب، مشروط به این است که تعریف سرویس مرکب به گونه‌ای باشد که اگر در هر جایی در طول اجرای سرویس مرکب خطا رخ داد، سرویس قابل خنثاکردن باشد. برخی از مدل‌های ارائه شده، برای رسیدن به چنین رفتاری، وجود سرویس خنثاکننده را برای تمامی سرویس‌ها فرض می‌گیرند و در واقع تعریف سرویس‌هایی با طریقه-ی بازیابی قطعی را مجاز نمی‌دانند. برای مثال WS-TXM فرض می‌کند تمامی سرویس‌های جزء سرویس مرکب، قابل خنثاکردن هستند. بدیهی است چنین پیش‌فرضی محدودیت‌های زیادی برای استفاده از سرویس‌هایی که، قابل خنثاکردن نیستند، برای موتور ترکیب ایجاد می‌کند.

در این بخش برای تعریف سرویس مرکب قابل خنثاشدن ابتدا چند مفهوم اولیه را تعریف می‌کنیم:

جریان کنترل[32]: ترتیب نسبی اجرای سرویس‌ها در یک جریان از تعریف سرویس مرکب است. می‌توانیم تعریف سرویس مرکب را به شکل یک گراف جهت‌دار در نظر بگیریم، که در آن دو نود ابتدا و انتهایی داریم و تمامی نودها روی مسیری از نود ابتدا به نود انتها قرار دارند. هر نود (بجز نودهای ابتدا و انتها) در واقع نماینده‌ی یک سرویس جزء در سرویس مرکب است و نود ابتدا و انتها نماینده‌ی شروع و پایان سرویس مرکب هستند. برای مثال شکل ۱ گراف تعریف یک سرویس مرکب است. یک جریان کنترلی در سرویس مرکب در واقع یک مسیر از نود ابتدا به نود انتها در گراف تعریف سرویس مرکب است. برای مثال (A,D,E,F) و (A,C,F) دو جریان کنترل در گراف شکل ۱ هستند. هر جریان کنترل یک ترتیب قرار گرفتن سرویس‌ها در جریان سرویس مرکب را نشان می‌دهد.

مجموعه‌ی جریان‌های کنترلی[33]: مجموعه‌ی تمامی جریان‌های کنترلی ممکن در تعریف یک سرویس مرکب است. این مجموعه تمامی حالاتی که در آن‌ها سرویس‌های مختلف شرکت‌کننده در سرویس مرکب، ممکن است پشت سر هم اجرا شوند را نشان می‌دهد. برای مثال، مجموعه‌ی جریان‌های کنترلی گراف شکل ۱ [(A,C,F),(B,F),(G),(A,D,E,F)] است.



شکل ۱. تعریف سرویس مرکب قابل خنثاشدن

حال با استفاده از مجموعه‌ی جریان‌های کنترلی سرویس مرکب، رفتار تراکنشی قابل خنثاشدن را برای سرویس مرکب تعریف می‌کنیم. همان‌طور که گفتیم در صورت خاتمه‌ی سرویسی با رفتار تراکنشی قطعی، اثراتش قابل خنثاکردن نیست، پس در تعریف یک سرویس مرکب تمامی سرویس‌هایی که ممکن است قبل از یک سرویس قطعی اجرا شوند، باید قابل خنثاکردن باشند. بنابراین می‌توان «سرویس مرکب قابل خنثاشدن» (یا سرویس مرکب با رفتار تراکنشی قابل خنثاشدن) را این‌گونه تعریف کرد:

سرویس مرکب قابل خنثاشدن: سرویس مرکبی که در تمامی جریان‌های کنترلی در مجموعه‌ی جریان‌های کنترلی آن، تمامی سرویس‌هایی که قبل از سرویس‌های با رفتار تراکنشی قطعی قرار دارند، رفتار تراکنشی غیرقطعی داشته باشند (قابل خنثاشدن یا بدون تأثیر).

برای مثال برای اینکه سرویس مرکب شکل ۱ قابل خنثاشدن باشد، تنها سرویس‌های G ،F و B می‌توانند قطعی باشند. توجه به این نکته ضروری است که، در چنین حالتی موتور اجرا باید، فراخوانی سرویس‌های قطعی را تا فراخوانی تمامی سرویس‌های دیگر به تأخیر بیاندازد. در واقع شرطی که در تعریف سرویس مرکب قابل خنثاشدن آمده است، سرویسی را که بالقوه قابل خنثاشدن است تعریف می‌کند، برای قابل خنثاشدن سرویس به شکل بالفعل، موتور اجرا باید اجرای سرویس‌های قطعی را تا قبل از اجرای سایر سرویس‌ها به تأخیر بیاندازد. یعنی بعد از خاتمه‌ی اجرای سرویس‌های A، C، D و E موتور اجرا، اجرای موازی سرویس‌های G، F و B را آغاز کند. خانه‌های خالی در جدول ۱ با استفاده از این تعریف برای هر سرویس مرکب پر می‌شوند.

بدیهی است حتی در این حالت نیز ممکن است در اجرای سرویس مرکب با شرایطی مواجه شویم که به خنثاکردن اثرات سرویس قطعی نیاز است. همان‌طور که گفتیم، موتور اجرا در مثال شکل ۱ سرویس‌های G، B و F را به شکل موازی و بعد از خاتمه‌ی سایر سرویس‌ها اجرا می‌کند، اما در هر حال این سرویس‌ها هم‌زمان خاتمه نمی‌یابند و ممکن است مثلاً بعد از خاتمه B و G، F دچار خطا شود. الگوریتم بازیابی از خطا باید این شرایط را نیز در نظر بگیرد.

۴- گسترش زبان OWL-S بر اساس مدل تراکنشی ارائه‌شده

برای توصیف سرویس‌های مرکب زبان‌های توصیف مختلفی ارائه شده‌است، که مهم‌ترین آن‌ها عبارتند از BPEL4WS[17]، OWL-S [18]، WSMO [19] و SWSF [20]. از میان این زبان‌ها زبان معنایی OWL-S را برای توصیف سرویس‌های مرکب انتخاب کردیم. OWL-S (که قبلاً تحت عنوان DAML-S شناخته شده بود) یک هستشناسی به زبان OWL است که به عنوان بخشی از پروژه DARPA Agent Markup Language Program و به منظور تعریف سرویس‌های وب معنایی ایجاد شده است[۱۸]. این زبان یکی از موفق‌ترین زبان‌های پیشنهاد شده برای توصیف سرویس‌ها است که توسط شرکت‌های خصوصی و مؤسسات تحقیقاتی مورد حمایت می‌باشد. این هستشناسی کشف اتوماتیک سرویس، صدا کردن سرویس‌ها، ترکیب، ارتباط بین آن‌ها وکنترل اجرای آن‌ها را ممکن می‌سازد. در قالب این زبان سه بخش زیر برای سرویس‌های وب مشخص می‌شوند: ۱) ServiceProfile: بیان می‌دارد که سرویس چه کاری انجام می‌دهد، ۲) ServiceModel: نحوه‌ی کار و به بیان دیگر چگونگی انجام سرویس را مشخص می‌نماید، ۳) ServiceGrounding: نحوه‌ی دسترسی به سرویس را معین و به عبارت دیگر چگونگی استفاده از سرویس را بیان می‌کند.

www.SID.ir

Archive of SID

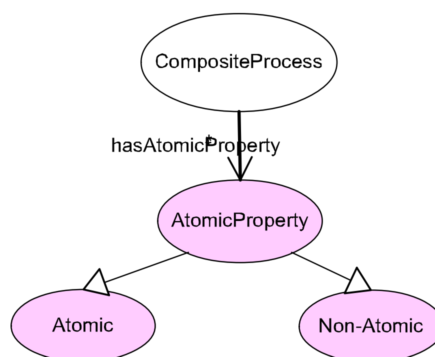
OWL-S را با توجه به قابلیت‌های پشتیبانی منطق (درستی‌یابی ترکیب)، پشتیبانی معیارهای کیفی سرویس (QoS)، پایداری و توسعه‌یافته‌گی و پشتیبانی توسط ابزارها برای مدل‌کردن مدل ارائه‌شده انتخاب کردیم. با توجه به این‌که این زبان، که توسط W3C ارائه شده‌است، مطرح‌ترین زبان معنایی است و در وب‌معنایی کاربردهای زیادی دارد، گسترش این زبان برای مدل‌کردن رفتارهای تراکنشی سرویس‌ها، که تا به حال انجام نشده‌است، کاربردهای زیادی دارد.

تعریف مدل رفتار تراکنشی شامل دو بخش اصلی‌است. بخش اول به نحوه‌ی تعریف یک تراکنش می‌پردازد و بخش دوم به تعریف رفتار تراکنشی برای هر یک از سرویس‌ها اختصاص دارد.

۴-۱- گسترش زبان OWL-S برای تعریف تراکنش

همانطور که گفتیم، بخش ServiceModel از هستشناسی OWL-S سرویس وب را به شکل یک فرآیند با جزئیات لازم مربوط به اجرای آن مدل می‌کند. این هستشناسی امکان تعریف سرویس‌های مرکب را از طریق کلاس CompositeProcess (یکی از زیرکلاس‌های کلاس Process) فراهم می‌کند. کلاس CompositeProcess از طریق ارتباط composedOf با کلاس ControlConstuct سرویس‌های تشکیل‌دهنده‌ی سرویس مرکب و نحوه‌ی اجرای آن‌ها (بر اساس نوع ساختار کنترلی) را مشخص می‌کند.

همان‌طور که در مدل رفتار تراکنشی در بخش قبل گفتیم، هر سرویس مرکب ممکن است رفتار تراکنشی داشته باشد یا نه، یا به عبارت دقیق‌تر اجرای آن ممکن است به شکل یک تراکنش باشد، یعنی از نظر تعریف‌کننده‌ی سرویس مرکب، ممکن است از این سرویس انتظار اجرای اتمیک داشته باشد و یا اینکه چنین انتظاری نداشته باشد. با توجه به اینکه بخش ProcessModel فرآیند واقعی اجرای سرویس را مدل می‌کند و در واقع موتور اجرا بر اساس تعریفی که این بخش از سرویس ارائه می‌کند، سرویس را اجرا می‌کند، برای مشخص‌کردن تعریف تراکنش، این بخش از هستشناسی OWL-S گسترش می‌یابد. بر همین اساس همان‌طور که در شکل ۲ می‌بینید در مدل گسترش‌داده‌شده، برای کلاس CompositeProcess با کلاس AtomicProperty ارتباط hasAtomicProperty را اضافه‌کرده‌ایم. کلاس AtomicProperty دو زیر کلاس Atomic و NonAtomic دارد که دارا بودن رفتار اتمیک را برای سرویس مرکب مشخص می‌کنند. همان‌طور که گفتیم هر سرویس مرکب می‌تواند خود بخشی از یک سرویس مرکب دیگر باشد و به این ترتیب برای هر بخش از سرویس مرکب که نوع رفتار تراکنشی خاصی انتظار می‌رود، می‌توان مشخصاً رفتار اتمیک یا غیر اتمیک را تعریف کرد. با توجه به نوع کلاس AtomicProperty موتور اجرا تشخیص‌می‌دهد که آیا باید در اجرای این سرویس مرکب محدودیت‌های اجرای اتمیک را در نظر بگیرد یا نه.



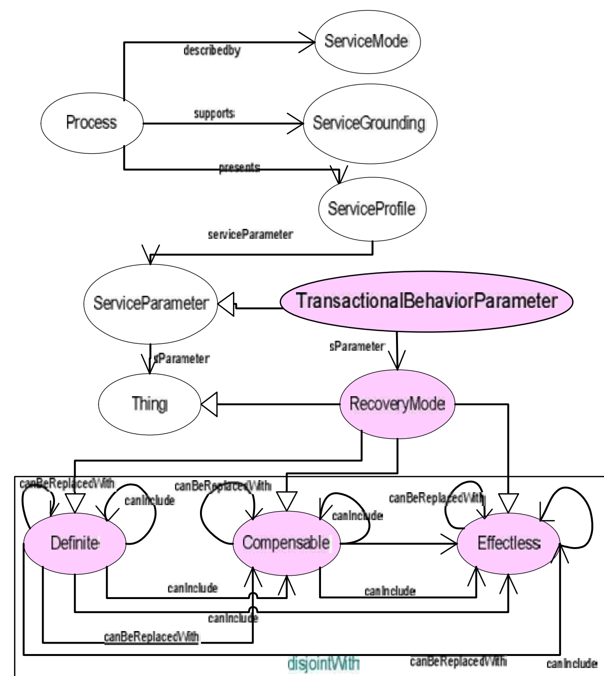
شکل ۲. گسترش زبان OWL-S برای تعریف تراکنش

۴-۲- گسترش زبان OWL-S برای تعریف رفتار تراکنشی هر سرویس

بخش ServiceProfile (یکی از سه بخش اصلی هستشناسی OWL-S) مکانیزمی برای توصیف سرویس در اختیار قرار می‌دهد. این توصیف به طور خاص بیش‌تر در بخش تبلیغات برای سرویس مورد استفاده قرار می‌گیرد و عملیات همتایابی براساس توصیفی که

www.SID.ir

در این قسمت از سرویس ارائه می‌شود، انجام می‌گیرد. هست‌شناسی ServiceProfile توسط ارتباط serviceParameter با کلاس ServiceParameter در ارتباط است. این کلاس پارامترهای مشخصه یک سرویس را مدل می‌کند، برای مثال، مشخصات کیفی سرویس نیز از این طریق قابل مدل‌کردن هستند. به همین دلیل برای مدل‌کردن مشخصه‌ی طریقه‌ی بازیابی از مکانیزم موجود در این قسمت یعنی از کلاس ServiceParameter استفاده کرده‌ایم. شکل 3 نحوه گسترش هست‌شناسی ServiceProfile را از نقطه گسترش کلاس ServiceParameter نشان می‌دهد، کلاس TransactionalBehaviorParameter را به عنوان زیر کلاس ServiceParameter به عنوان نوعی مشخصه‌ی سرویس تعریف کرده ایم. به علاوه RecoveryMode را به صورت زیر کلاس Thing و به عنوان کلاسی که برد[۳۴] رابطه sParameter است تعریف کرده‌ایم. انواع رفتارهای تراکنشی تعریف شده در مدل تراکنشی ارائه شده (Affectless، Compensable و Definite) را به‌عنوان زیرکلاس‌های کلاس RecoveryMode تعریف می‌کنیم.



شکل ۳. گسترش OWL-S برای تعریف رفتار تراکنشی هرسرویس

بر اساس مدل تراکنشی ارائه‌شده طریقه‌ی بازیابی یک سرویس مرکب براساس طریقه‌های بازیابی سرویس‌های تشکیل‌دهنده و طبق جدول ۱ تعیین می‌شود. برای مدل‌کردن این جدول در هست‌شناسی گسترش داده شده، ارتباط canInclude را با دامنه و برد RecoveryMode تعریف می‌کنیم. این ارتباط به این معنی است که کدام نوع تراکنش‌ها (با کدام انواع طریقه‌ی بازیابی) می‌توانند در تراکنشی از نوعی دیگر قرار داشته باشند. برای مثال این ارتباط همان‌طور که در شکل مشخص است، نشان‌می‌دهد که تراکنش با طریقه بازیابی Definite می‌تواند شامل تمامی انواع تراکنش‌های دیگر باشد و تراکنش با طریقه بازیابی Affectless نیز می‌تواند در تمامی تراکنش‌های دیگر (بدون تأثیر) قرار بگیرد.

در مدل تراکنشی ارائه‌شده دیدیم که انواع طریقه‌های بازیابی قابل ترکیب نیستند، این رفتار نیز، از طریق ارتباط disjointWith تعریف شده‌است.

بر اساس مدل رفتار تراکنشی ارائه شده، برای طریقه‌ی بازیابی قابل خنثاشدن، مشخصه‌های زیر باید مدل شوند (شکل ۴):

۱. سرویس خنثاکننده: مهم‌ترین مشخصه برای تراکنش با طریقه‌ی بازیابی قابل خنثاشدن، سرویس یا در واقع تراکنش خنثاکننده است. سرویس خنثاکننده از طریق کلاس Service که با ارتباط CompensableService با کلاس Compensable در ارتباط است مشخص می‌شود. برد این ارتباط نمونه‌های کلاس Service از هست‌شناسی OWL-S است.

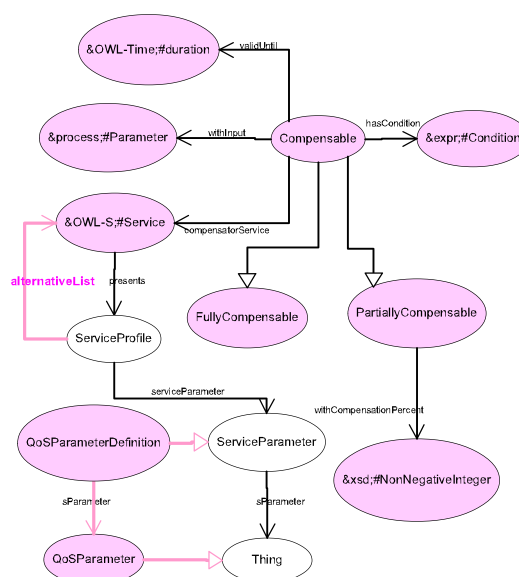
۲. ورودی‌های سرویس خنثاکننده: سرویس خنثاکننده قطعاً پارامترهایی به عنوان ورودی نیاز دارد که باید مشخص شود از کجا باید فراهم شوند. این ورودی‌ها در توصیف سرویس خنثاکننده آورده می‌شوند، اما در این بخش به طور خاص نیز مشخص می‌شوند. همان‌طور که گفتیم ورودی‌های سرویس خنثاکننده را به شکل نمونه‌هایی از کلاس Parameter از هستشناسی Process (بخشی از هستشناسی OWL-S) مدل می‌کنیم. بدیهی است که این نمونه‌ها باید زیر مجموعه‌ی پارامترهای سرویس اولیه باشند و در این‌جا تنها به آن‌ها اشاره می‌کنیم (کلاس Parameter زیر کلاسِ کلاس‌های Input، Output و Result است، در نتیجه تمامی حالت‌های ورودی‌های ممکن سرویس خنثاکننده را پوشش می‌دهد).

۳. پیش‌شرطها: شرایط یا به عبارت دیگر پیش‌شرطهای سرویس خنثاکننده مشابه پیش‌شرطهای Service در هستشناسی OWL-S مدل می‌شوند، یعنی به شکل فرمول‌های منطقی و با استفاده از هستشناسی Expression (این هستشناسی به عنوان بخشی از پروژه OWL-S توسط W3C ارائه شده است [۲۱]). پیش‌شرط اجرای سرویس خنثاکننده با ارتباط hasCondition با کلاس Compensable مشخص می‌شود.

۴. هزینه: به‌عنوان یکی از معیارهای کیفی و از طریق مکانیزم serviceParameter که در OWL-S برای مدل کردن معیارهای کیفی در نظر گرفته شده‌است، مدل‌می‌شود.

۵. محدویت زمانی: محدویت زمانی برای اجرای سرویس خنثاکننده در واقع از نوع یک بازه‌ی زمانی است، مثلاً ۱ روز و ۲ ساعت، که سرویس خنثاکننده به این مقدار بعد از اجرای سرویس اولیه امکان خنثاکردن اثراتش را دارد. این بازه‌ی زمانی با ارتباط validUntil که بردش نمونه‌هایی از کلاس duration در هستشناسی OWL-Time است، مدل می‌شود [۲۲]. این هست-شناسی توسط W3C ارائه شده است، تمامی جنبه‌های زمانی را مدل می‌کند، و روابط معنایی مختلف مورد نیاز برای مدل کردن زمان را نیز به شکل صوری تعریف می‌کند. به‌علاوه این هستشناسی در سناریوهای متعددی در تعاریف سرویس‌های معنایی براساس OWL-S به عنوان نمونه به‌کاررفته است.

همان‌طور که در مدل رفتار تراکنشی گفتیم، طریقه‌ی بازیابی قابل خنثاشدن خود به دو دسته Fully Compensable و Partially Compensable رده بندی می‌شود. این دسته‌بندی، همان‌طور که در شکل ۴ مشخص است، به شکل زیر کلاس‌های کلاس Compensable به همین نام مدل می‌شود. کلاس FullyCompensable مشخص‌کننده‌ی طریقه‌ی بازیابی سرویس‌هایی است که تأثیر (effect) سرویس خنثاکننده‌ی آن‌ها دقیقاً برعکس سرویس اصلی است. برای سرویس‌های قابل خنثاشدن ناقص، میزان قابلیت خنثاسازی سرویس خنثاکننده به شکل درصد توصیف می‌شود. این درصد با ارتباط withCompensationPercent مدل می‌شود.

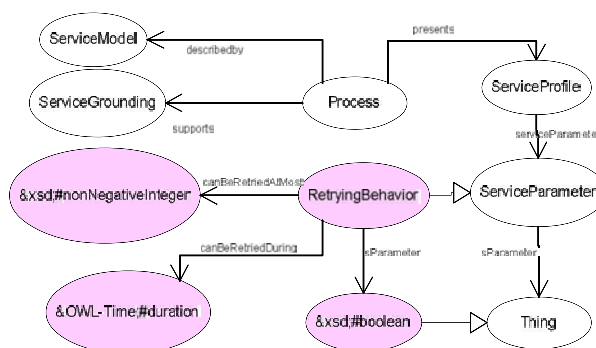


شکل ۴. طریقه‌ی بازیابی قابل خنثاشدن

شکل ۵ نحوه گسترش هستشناسی ServiceProfile را از نقطهی گسترش کلاس ServiceParameter برای مدل کردن مشخصهی قابلیت تلاش مجدد نشان میدهد. کلاس RetryingBehavior را به عنوان زیر کلاس ServiceParameter به عنوان نوعی مشخصهی سرویس تعریف کردهایم. برد ارتباط sParameter نمونههایی از نوع Boolean هستند.

بر اساس مدل رفتار تراکنشی برای قابلیت فراخوانی مجدد دو مشخصه مدل میشوند (شکل ۵):

۱. بیشینهی تعداد دفعاتی که میتوان سرویس را فراخوانی مجدد کرد، توسط ارتباط canBeRetriedAtMost بین کلاس RetryingBehavior و NonNegativeInteger تعریف میشود.
۲. محدودیت زمانی: محدودیت زمانی برای اجرای مجدد سرویس تا اجرای موفقیت آمیز سرویس، را با ارتباط canBeRetriedDuring بین کلاسهای RetryingBehavior و duration تعریف میکنیم. برای مدل کردن بازهی زمانی که طی آن فراخوانی مجدد سرویس از نظر تأمینکنندهی آن مجاز است، از هستشناسی OWL-Time که توسط W3C ارائه شده است استفاده میکنیم [۲۲].



شکل ۵. قابلیت تلاش مجدد

۵- الگوریتم بازیابی از خطا

الگوریتم ارائهشده در شکل ۶ پس از بروز خطا در اجرای یکی از سرویسهای تشکیلدهندهی سرویس مرکب فراخوانی میشود. فرض بر این است که تعریف سرویس مرکب با تعریف سرویس مرکب قابل خنثاکردن مطابقت دارد و تعریف آن بر اساس تعریف خنثاشدن برای سرویس مرکب سازگار است (طبق تعریفی که در بخش ۳-۲ آمدهاست).

به طور کلی برای مدیریت و بازیابی خطای رخ داده، دو رویکرد کلی وجود دارد. رویکرد اول رویکرد حرکت به جلو[۳۵] است. در این رویکرد سعی بر ادامه اجرای سرویس مرکب از طریق فراخوانی مجدد سرویس میشود. در صورتیکه ادامه اجرا از این طریق ممکن نباشد، با استفاده از رویکرد حرکت به عقب[۳۶] سعی میکند تا آثار و نتایج سرویسهایی که تا به حال اجرا شدهاند را تا حد امکان خنثاکند، به گونهای که بهتوان ادعا کرد اجرای سرویس مرکب هیچگاه آغاز نشدهاست. در الگوریتم ارائهشده برای مدیریت خطای رخ داده، از هر دو رویکرد حرکت به جلو و حرکت به عقب استفادهمیشود.

در این بخش ابتدا جزئیات الگوریتم ارائهشده را بررسی میکنیم و در ادامه به تحلیل الگوریتم میپردازیم.

۵-۱- الگوریتم بازیابی از خطا

همانطور که میبینید این الگوریتم به عنوان ورودی سرویسی که دچار خطا شده است را دریافت میکند. در لحظهای که سرویس دچار خطا شده است، موتور اجرا ممکن است به شکل موازی درحال اجرای سرویسهای دیگری نیز باشد. با توجه به اینکه ممکن است بخش حرکت به جلو نتواند اجرای سرویس را ادامه دهد و در نهایت نیاز به حرکت به عقب میباشد، برای کمینهکردن هزینههای مربوط به خنثاکردن و یا پایان اجرای سرویسهای در حال اجرایی که قابل خنثاکردن نیستند، در ابتدای الگوریتم، لیست سرویسهای در حال اجرا از موتور اجرا گرفته میشود و اجرای آنها با ارسال پیام لغو[۳۷] به ارائهکننده سرویس، متوقف میشود. در

ادامه ابتدا بخش حرکت به جلو اجرا می‌شود و در صورت عدم موفقیت این قسمت در جایگزینی سرویس و ادامه اجرا، بخش حرکت به عقب اثرات سرویس‌های اجراشده‌ی خاتمه یافته را تا حد امکان خنثا می‌کند:

حرکت به جلو: همان‌طور که در شکل ۶ می‌بینید، ابتدا قابلیت سرویس برای اجرای مجدد چک می‌شود و در صورتی‌که سرویس قابل تلاش مجدد باشد، محدودیت زمانی برای اجرای سرویس بررسی می‌شود. همان‌طور که در مدل تراکنشی گفتیم ممکن است اجرای مجدد سرویس تا مدت زمان مشخصی مجاز باشد، و این محدودیت در این مرحله چک می‌شود. در صورتی‌که بازه زمانی مورد نظر (canBeRetriedDuring) نگذشته باشد، سرویس به تعداد دفعاتی که می‌توان مجدداً سرویس را اجرا کرد (canBeRetriedAtMost)، و تا اجرای موفقیت‌آمیز، فراخوانی می‌شود. در صورتی‌که فراخوانی مجدد سرویس موفقیت‌آمیز باشد، درواقع الگوریتم توانسته است در حرکت به جلو موفق شود و اجرای سرویس مرکب از همان نقطه‌ای که متوقف شده‌بود، باید ازسر گرفته‌شود. الگوریتم خاتمه می‌یابد و پیام «ادامه‌ی اجرای سرویس مرکب» به موتور اجرا ارسال می‌گردد.

حرکت به عقب: در صورتی‌که ادامه‌ی اجرای سرویس مرکب موفقیت‌آمیز نباشد، تأثیرات سرویس‌هایی که تا به حال در قالب این سرویس مرکب اجرا شده‌اند باید خنثاشود. برای این منظور، الگوریتم در این مرحله، لیست تمامی سرویس‌هایی که اجرا شده‌اند را دریافت می‌کند (compensationList). با استفاده از Log کردن، می‌توان لیست سرویس‌های اجرا شده را در اختیار داشت. برای هر یک از سرویس‌های اجرا شده (و خاتمه یافته)، بر اساس طریقه‌ی بازیابی سرویس، تصمیمات مختلفی اتخاذ می‌شود (switch-case). برای هر یک از انواع طریقه‌های بازیابی به شکل زیر عمل می‌شود.

۱. **طریقه‌ی بازیابی بدون تأثیر**: همان‌طور که گفتیم، این طریقه‌ی بازیابی مربوط به سرویسی است که تأثیراتی که نیاز به خنثا-کردن داشته باشد در وضعیت جهان ایجاد نمی‌کند، در نتیجه برای خنثاکردن چنین سرویسی هیچ عملیات خاصی انجام نمی‌شود.

۲. **طریقه‌ی بازیابی قطعی**: این طریقه‌ی بازیابی مربوط به سرویس‌هایی است که تأثیرات‌شان به هیچ عنوان قابل خنثاکردن نیستند، در نتیجه برای خنثاکردن چنین سرویسی عملیات اتوماتیکی قابل انجام نیست و وضعیت پیش آمده برای انجام اقدامات لازم به مدیر سیستم اطلاع داده می‌شود. توجه به این نکته لازم است که فرض بر این است که تعریف سرویس مرکب، قابل خنثا کردن است (با تعریفی که برای سرویس مرکب قابل خنثاکردن ارائه کردیم)، در نتیجه تنها در شرایطی که چند سرویس قطعی در انتهای جریان‌های کنترلی مختلف موازی قرار دارند ممکن است سرویس قطعی قبل از خاتمه سرویس مرکب خاتمه یابد. البته در نظرگرفتن این حالت، باعث می‌شود که الگوریتم ارائه شده برای پشتیبانی سایر حالت‌ها و تعاریف سرویس‌های مرکب نیز صحیح عمل کند.

   به‌علاوه ممکن است در شرایطی که سرویس‌های کاندید در روال ترکیب خیلی محدود باشند، موتور ترکیب ترجیح دهد که در مورد شرط قابل خنثاکردن بودن برای سرویس مرکب کمی تخفیف دهد و سرویسی قطعی را در محلی غیر مجاز (از نظر تعریف سرویس مرکب قابل خنثاکردن) به کار ببرد.

۳. **طریقه‌ی بازیابی قابل خنثاکردن کامل**: در این حالت می‌دانیم که تأثیرات سرویس کاملاً قابل خنثاکردن است. تمامی سرویس‌های خنثاکننده‌ی ارائه شده برای سرویس را درنظر می‌گیریم و محدودیت‌های مختلف را به ترتیب اعمال می‌کنیم. ابتدا سرویس‌های خنثاکننده‌ای که بازه‌ی زمانی مجازشان گذشته است را فیلتر می‌کنیم. در مرحله بعد هزینه‌ی سرویس خنثاکننده را در نظر می‌گیریم و سرویس‌هایی که از لحاظ هزینه با پروفایل کاربر مطابقت دارند و با توجه به هزینه‌ی کل سرویس مرکب هنوز از سقف هزینه‌ی قابل پرداخت کاربر تجاوز نمی‌کنند را انتخاب می‌کنیم. در مرحله بعدی پیش‌شرط‌های اجرای سرویس خنثاکننده چک می‌شوند و سرویس‌های خنثاکننده‌ای که پیش شرط‌هایشان برقرار نیست نیز فیلتر می‌شوند.

توجه کنید که بررسی پیش‌شرط‌ها پیچیده‌تر از بررسی هزینه و محدودیت زمانی است و در نتیجه برای بهبود کارایی الگوریتم به عنوان آخرین شرط چک می‌شود، بعد از اینکه لیست سرویس‌های خنثاکننده را با توجه به تمام محدودیت‌ها فیلتر کردیم، سرویس خنثا کننده با کم‌ترین هزینه را انتخاب کرده و اجرا می‌کنیم. درصورتی‌که هیچ سرویس خنثاکننده‌ای بعد از اعمال فیلترها باقی نماند، در واقع الگوریتم در خنثاکردن اثرات سرویس ناموفق است و باید شرایط را برای اقدامات لازم به مدیر سیستم اطلاع داد.

www.SID.ir

Archive of SID

۴. **طریقه‌ی بازیابی قابل خنثاکردن ناقص**: در این حالت نیز مانند حالت قبلی، سرویس‌ها را بر اساس محدودیت‌های هزینه، بازه زمانی مجاز و پیش‌شرط‌های سرویس فیلتر می‌کنیم. در نهایت از بین سرویس‌های خنثاکننده‌ی باقی مانده، سرویسی که بیش‌ترین درصد خنثاکردن را دارد (سرویسی که بیش‌تر از سایرین می‌تواند اثرات را خنثا کند) انتخاب و اجرا می‌شود.

این الگوریتم در دو حالت خاتمه می‌یابد، یا در قسمت حرکت به جلو با اجراهای مجدد، سرویس موفقیت‌آمیز اجرا می‌شود و یا این‌که تا حد امکان با حرکت به عقب آثار و نتایج سرویس‌هایی که در قالب این سرویس مرکب اجرا شده‌اند خنثا می‌شوند.

۵-۲- تحلیل الگوریتم

در این بخش تحلیل الگوریتم ارائه شده برای بازیابی از خطا از نظر زمانی ارائه می‌شود. در تحلیل الگوریتم پارامترهای ورودی زیر در نظر گرفته می‌شوند:

- تعداد کل سرویس‌های شرکت‌کننده در سرویس مرکب را m در نظر می‌گیریم.
- بیشینه تعداد دفعات تلاش مجدد برای اجرای سرویس نیز r در نظر گرفته می شود.
- بیشینه تعداد سرویس‌های خنثاکننده برای یک سرویس را نیز k در نظر می گیریم.
- بیشینه تعداد پیش‌شرط‌های سرویس‌های خنثاکننده‌ی یک سرویس p است.

تحلیل زمانی الگوریتم به شرح زیر است:

- خاتمه سرویس‌های در حال اجرا: بیشینه تعداد سرویس‌های در حال اجرا m است و هزینه‌ی خاتمه معادل ارسال پیام لغو به هر یک از سرویس‌هاست. پس هزینه‌ی خاتمه‌ی کل سرویس‌ها از O(m) است.
- تلاش مجدد برای اجرای سرویس: هزینه‌ی زمانی این بخش حداکثر برابر بیشینه تعداد دفعات تلاش مجدد برای اجرای سرویس است، یعنی در بدترین حالت، اگر سرویسی که بیش‌ترین تعداد دفعات تلاش مجدد را دارد، دچار خطا شود، هزینه‌ی این قسمت الگوریتم O(r) است.
- بخش حرکت به عقب: در این قسمت برای تمامی سرویس‌هایی که تا به حال اجرا شده اند (CompensationList) عملیات‌های لازم انجام می‌شود، که بیشینه‌ی تعداد این سرویس‌ها m است. عملیاتی که برای هر سرویس انجام می‌شود، در حالت‌های Effectless و Definite از O(1) است و در هر دو نوع FullyCompensable و PartiallyCompensable از O(p.k) است. در این قسمت لیستی به طول k طی مراحلی فیلتر می‌شود. به جز filterListAccordingToPreconditions که از O(p.k) است، بقیه از O(k) هستند، پس کل هزینه‌ی زمانی این بخش از O(p.k) است. در انتها نیز یک گرفتن بیشینه و کمینه روی همین لیست را داریم، که از O(k) است.

  به این ترتیب کل هزینه ی بخش حرکت به عقب در بدترین حالت از O(m.p.k) است.

بنابر آن‌چه گفتیم، هزینه‌ی کل الگوریتم بازیابی از خطا در بدترین حالت برابر O(m.p.k)+O(r)+O(m) است، که در واقع باید O(m.p.k)+O(r) در نظر گرفته‌شود.

۶- نتیجه

رویارویی با خطاهای زمان اجرا از مهم‌ترین مسائل مطرح در حوزه‌ی اجرای سرویس‌های مرکب است. پشتیبانی تراکنش‌ها در محیط سرویس‌های وب، باید با توجه به مشخصات خاص این محیط صورت بگیرد و در نتیجه استفاده از تراکنش‌های کلاسیک ACID در این حوزه ممکن نیست. در این مقاله مدلی برای پشتیبانی تراکنش‌ها در سرویس‌های مرکب ارائه شده‌است. در این مدل که از ایده‌ی خنثاکردن استفاده می‌کند، رفتارهای تراکنشی مختلف، متناسب با محیط ناهمگن وب، با جزئیات کامل تعریف شده‌اند، که در کارهای مشابه، تعریف دقیق و همه جانبه‌ی رفتارهای تراکنشی نیامده‌است. به‌عنوان بخش دیگری از این مدل، تعریف سرویس مرکب قابل خنثاشدن ارائه شده‌است. در واقع تعریفی که از سرویس مرکب برای اجرا به موتور اجرا داده می‌شود، باید با این تعریف سازگار باشد.

www.SID.ir

هست شناسی OWL-S نیز بر اساس مدل رفتار تراکنشی ارائه‌شده گسترش داده شده‌است و به‌این‌ترتیب مدل ارائه‌شده به شکل یک هست‌شناسی مدل شده‌است. در هیچ‌یک از کارهای ارائه‌شده، این ارتباط بین مدل رفتار تراکنشی با تعریف سرویس مرکب نیامده‌است. با توجه به گسترش استفاده از زبان OWL-S در وب معنایی، این بخش کاربردهای زیادی دارد.

به‌علاوه الگوریتمی برای بازیابی از خطا ارائه شده‌است، که از هر دو رویکرد حرکت به جلو و حرکت به عقب استفاده می‌کند. این الگوریتم بر اساس مدل ارائه‌شده و با درنظرگرفتن تمامی محدودیت‌ها ابتدا سعی در ادامه‌ی اجرا از طریق فراخوانی مجدد می‌کند (رویکرد حرکت به جلو). درصورت عدم امکان ادامه‌ی اجرا، اثرات سرویس‌های اجراشده قبل از بروز خطا را تاحدامکان خنثا می‌کند (رویکرد حرکت به عقب).

```
Algorithm FailureRecovery
Input: failed service
{// Abort executing services;
Get list of all currently executing services from Execution Engine and abort them.
// Forward Mechanism
if failed service is Retriable {
        if failedService canBeRetriedDuring is not passed
                Re-execute failedService at most (failedService.canBeRetriedAtMost) times until
                its successful execution;
        if failedService executed successfully
                return(resume-composite-service-execution);}
// Backward Mechanism
compensationList := getAllExecutedServices();
for each service in compensationList do
{switch(service.recoveryMode) // decide according to recovery mode of failed service
        case(Effectless): break;
        case(Definite): Notify Administrator by logging definite service specification;
        case(Fully Compensable):{
                compensatorsList := service.compensatorService;
                compensatorsList := filterListAccordingToTimeLimit(compensatorList,
                service.validUntil);
                compensatorsList := filterListAccordingToCostLimit(compensatorList);
                compensatorsList := filterListAccordingToPreConditions(compensatorList,
                service.hasCondition);
                if compensatorList is not empty
                        execute the cheapest compensator service;
                else
                        notify Administrator by logging;
        case(Partially Compensable):{
                compensatorsList := service.compensatorService;
                compensatorsList := filterListAccordingToTimeLimit(compensatorList,
                service.validUntil);
                compensatorsList := filterListAccordingToTimeCost(compensatorList,
                service.withCost);
                compensatorsList := filterListAccordingToPreConditions(compensatorList,
                service.hasCondition);
                if compensatorList is not empty
                        execute the compensator service with the most compensation percent;
        }
}
return();}
```

شکل ۶. الگوریتم بازیابی از خطا

مراجع

[1] J. Gray. The Transaction Concept: Virtues and Limitations. In *Proceedings of Very Large Data Bases*, pages 179–201, 1981.

[2] Web Services Transaction (WS-Transaction), BEA Systems, International Business Machines Corporation, Microsoft Corporation, Inc., http://www.ibm.com/developerworks/library/wstranspec.

[3] D. Langworthy and al. Web Services Atomic Transaction (WS-AtomicTransaction). 2004.
[4] D. Langworthy and al. Web Services Business Activity Framework (WS-BusinessActivity). 2004.
[5] D. Langworthy and al. Web Services Coordination (WS-Coordination). 2004.
[6] D. Bunting, M.C.O. Hurley, M. Little., J. Mischkinsky, E. Newcomer, Webber, J. and Swenson, K. (2003b) "*Web Services Composite Application Framework (WS-CAF) Ver1.0*", July 28, 2003.
[7] S. Bhiri, O. Perrin, and C. Godart, "Ensuring required failure atomicity of composite web services." in *WWW*, 2005, pp. 138–147.
[8] OASIS Committee Specification. Business Transaction Protocol, Version 1.0 (June 2002).
[9] D. Biswas. Compensation in the world of web services composition. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, Revised Selected Papers, volume 3387 of Lecture Notes in Computer Science, pages 69–80. Springer-Verlag, July 2004.
[10] Compensation with Dependency in Web Services Composition, Lili Lin; Fangfang Liu Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on Volume , Issue , 22-26 Aug. 2005 Page(s): 183 – 188, Digital Object Identifier 10.1109/NWESP.2005.24
[11] T. Mikalsen, S. Tai and I. Rouvellou, Transactional Attitudes: Reliable Composition of Autonomous Web Services. In Dependable Systems and Networks Conference, (Maryland, USA, 2002).
[12] S. Bhiri, C. Godart, O. Perrin, Reliable Web Services Composition using a Transactional Approach, Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005, 15-21.
[13] B.A. Schmit, S. Dustdar, Towards transactional Web services, Seventh IEEE International Conference on E-Commerce Technology, 2005. 2005, 12- 20.
[14] Y. Ren, Q. Wu, Y. Jia, J. Guan and W. Han, Transactional business coordination and Failure Recovery for Web Services Composition, GCC, 2004, pp:26-33.
[15] B. Limthanmaphon and Y. Zhang. Web service composition transaction management. In Proceedings of the Fifteenth Australian Database Conference, volume 27 of Conferences in Research and Practice in Information Technology, pages 171–179.Australian Computer Society, Jan. 2004.
[16] Concepts and applications of multilevel transactions and open nested transactions, In A Elmagamid, editor, Database Transaction Models for Advanced Applications, Morgan-Kaufmann.
[17] "*Business Process Execution Language for Web Services, version 1.1*", http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf
[18] D. Martin et al., "OWL-S: Semantic Markup for Web Services". [online]. Available: http://www.w3.org/Submission/2004/SUBM-OWLS-20041122
[19] J. Bruijn, "Web Service Modeling Ontology (WSMO)", June 2005, [online]. Available: http://www.w3.org/Submission/WSMO
[20] SWSF [f-1.15] S. Battle et al., "Semantic Web Services Framework (SWSF) Overview", September 2005, [online]. Available: http://www.w3.org/Submission/SWSF
[21] http://www.daml.org/services/owl-s/1.1/generic/Expression.owl
[22] http://www.w3.org/TR/2006/WD-owl-time-20060927/

زیرنویس‌ها

[1] Atomicity
[2] Consistency
[3] Isolation
[4] Durability
[5] Loosely Coupled
[6] Long Running Transaction
[7] Rollback
[8] Context
[9] Atomic Transaction
[10] Interaction
[11] Two-phase Commit
[12] Volatile 2PC
[13] Durable 2PC
[14] Business Activity
[15] Interoperability
[16] Activity

[17] Accepted Termination States
[18] Retriable
[19] Compensable
[20] Pivot
[21] State/Transition Diagram
[22] Business Transaction Protocol
[23] Cohesive
[24] Atomic Property
[25] Retriable Mode
[26] Recovery Mode
[27] Affectless
[28] Compensable
[29] Definite
[30] Fully Compensable
[31] Partially Compensable
[32] Control Flow
[33] Control Flow Set
[34] Range
[35] Forward Error Recovery
[36] Backward Error Recovery
[37] Cancel

www.SID.ir