تحمل پذیری خطا در یک سیستم مدیریت توزیع:

مطالعه موردی

چکیده

این مطالعه مهم‌ترین مفهوم فراگرفته شده از اجرای یک سیستم مدیریت ارتباط از راه دور توزیع‌شده را (DTM ها)، که یک سیستم ارتباط صوتی شبکه‌شده را کنترل می‌کند بیان می‌کند. الزامات اساسی مورد نیاز برای DTM ها تحمل‌پذیری خطا در برابر شکست‌های سایت یا شبکه، امنیت کاربردی و قابلیت اعتماد ماندگار است. به‌منظور ارائه توزیع و ماندگاری هر دو مفهوم شفافیت و مقاومت در تحمل‌پذیری خطا، معماری دو لایه الگوریتم تکرار را معرفی می‌کنیم. درمیان مفاهیم فرارگرفته شده: مهندسی نرم‌افزار براساس مولفه‌ها، با سربار اولیه قابل‌توجهی همراه است اما در دراز مدت باارزش است. سرویس تحمل‌پذیری در برابر خطا یکی از نیازهای کلیدی برای توزیع خرابی امن است. دانه دانه شدگی منطقی برای کنترل مقاومت و همزمانی کل شی است. تکرار ناهمزمان در لایه پایگاه داده نسبت به تکرار همزمان در سطح بالاتری از نظر استحکام و قوام قرار دارد؛ مقاومت نیمه‌ساختاریافته با XML دارای اشکالاتی در مقاومت، عملکرد و راحتی؛ در مقابل مدل شی دارد، ساختار سلسله مراتبی قوی‌تر و امکان‌پذیرتر است. یک موتور پرس‌وجو به وسیله‌ای برای انتقال از طریق مدل شی اتلاق می‌شود؛ در نهایت انتشار عملیات حذف در مدل شی‌گرایی پیچیده‌تر م شود. بنا به مطالب فرا گرفته شده ما قادر به ارائه پلت‌فرم توزیع دردسترس برای سیستم‌های شی مقاوم هستیم.

1. مفاد سیستم

دردسترس بودن بالا نه تنها برای ایمنی بحرانی سیستم ارتباط صوتی شبکه نیاز است (VCS) بلکه برای کنترل سیستم‌های مدیریت نیز ضروری است. در این مطالعه، شبکه‌های VCS توسط یک سیستم مدیریت ارتباط از راه دور توزیع‌شده (DTM ها) همانند شکل 1 مدیریت می‌شوند. سرورهای متعدد DTM از طریق یک شبکه گسترده WAN به هم متصل هستند و هر سرور DTM، سیستم‌های VCS مرتبط با آن را تنظیم، کنترل و نظارت می‌کند. هر DTM پارامتر VCS را با استفاده از یک مدل شی‌گرا نگه‌داری می‌کند.

به‌طورکلی پیروی از اصول طراحی مهندسی نرم‌افزار براساس اجزا (CBSE) [6] مانند انسجام دائم اجزاء قوی و استانداردهای با تعامل خوب، شرایط زیر را می‌طلبد که مربوط به دردسترس بودن است:

تحمل‌پذیری خطا: سیستم در برابر خطاها تحمل‌پذیر است. این کار از طریق تکرار شی در سایت‌های دیگر امکان‌پذیر است.

- همه اشیاء نیاز دارند تا در تمام مواقع حتی در حضور سناریوهای دلخواه سیستم جهت خواندن دردسترس باشند.
- برای دسترسی جهت نوشتن کافی است

✓ اگر اشیاء در تمام سایت‌ها در طول وضعیت سالم سیستم در دسترس هستند.

✓ اگر اشیاء در یک سایت خاص در طول وضعیت تخریب سیستم (که ما سایت را فراخوانی می‌کنیم) دردسترس هستند.

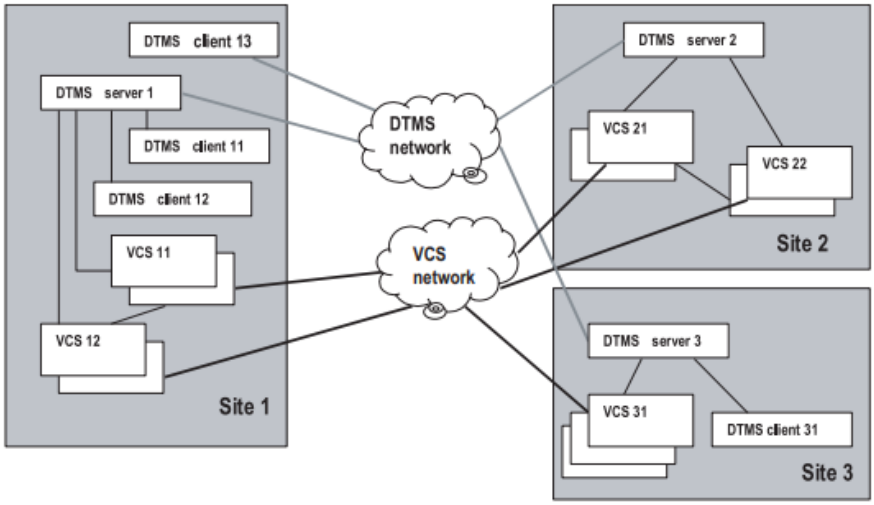
علاوه براین، خود چارچوب به‌طورکامل توزیع‌شده است. چارچوب منحصربه فرد تک جزء نباید به‌منظور جلوگیری از هرگونه شکستی در سراسر کل سیستم وجود داشته باشد.

تراکنش: اشیاء باید قادر به شرکت در تراکنش باشند. به‌طور معمول، نوشتن کوچک و خواندن بزرگ تراکنش‌ها اغلب نیاز به اجرا دارد.

تداوم: داده‌ی شی مقاوم، باید در منبعی پایدار ذخیره شود.

شفافیت: مشتریان نباید از مسائل توزیع‌شدگی، تکرار و تداوم آگاه باشند، در نتیجه شفاف برای مشتریان وجود دارد.

در مطالعه ما، هر دو مورد توزیع‌شدگی و مقاومت یا تداوم، باید شفاف و مقاوم در برابر تحمل‌پذیری خطا باشند، که تحمل‌پذیری خطا توانایی یک سیستم برای ادامه عملکرد در مواقعی است که شکست هنوز ترمیم نشده و یا حتی غیر قابل تشخیص است [7].



شکل 1. بررسی اجمالی DMTS.

از آنجا که تحمل‌پذیری خطا نیاز به گنجانده شدن در معماری سیستم دارد، ما شی حفظ شده را، به‌منظور ارائه اطلاعات اضافی برای جایگزینی پویای هر شی شکست خورده‌ای به دیگر سایت‌ها تکرار می‌کنیم. در بهترین حالت، برنامه قابل دسترس خواهد بود اگر هر سایتی در دسترس باشد، زیرا درحال حاضر اشیاء را می‌توان در هر سایتی ذخیره کرد.

تکرار به‌عنوان وسیله‌ای برای ارائه تحمل‌پذیری خطا مناسب است و مجموعه‌ای از پروتکل‌های تکرار وجود دارد. چگونه تا به حال، به‌کارگیری تکرار نیازمند مدیریت شی تکرار شده و معرفی نیازمندی‌های جدید به شفافیت تکرار بود. علاوه بر این، اگر تداوم و توزیع نرم‌افزار سیستم در یک سیستم شی‌گرا متمرکز باشد، مدیریت فیزیکی متعدد به مفاهیم متعارف خود از چنین سیستمی که معمولا برای در یک نسخه واحد طراحی شده است بستگی دارد:

- حفظ ثبات در میان اشیاء منطقی
- دسترسی همزمان به اشیاء منطقی
- دسترسی تراکنشی به اشیاء منطقی

• به دست گیری هویت شی، که در میان اشیاء منطقی منحصر به فرد است

• به دست گیری هویت شی، که در میان اشیاء منطقی منحصر به فرد است

سهم این مطالعه، توصیف منطق اصلی برای تصمیم‌گیری‌های کلیدی و استنتاج کلی است، که از نظر ما، می‌توان از تجارب به دست آورد.

مهمترین یافته‌ها عبارتند از:

• توزیع و تکرار نیاز به نصب معماری دو لایه دارد. به این ترتیب، مکانیسم توزیع معمولی را می‌توان بدون تغییر مستقر کرد.

• سرویس نام‌گذاری خرابی امن یک شرط کلیدی برای توزیع شکست امن است.

• عملکرد موردنیاز برای اطمینان از پیچیدگی سیستم سخت است.

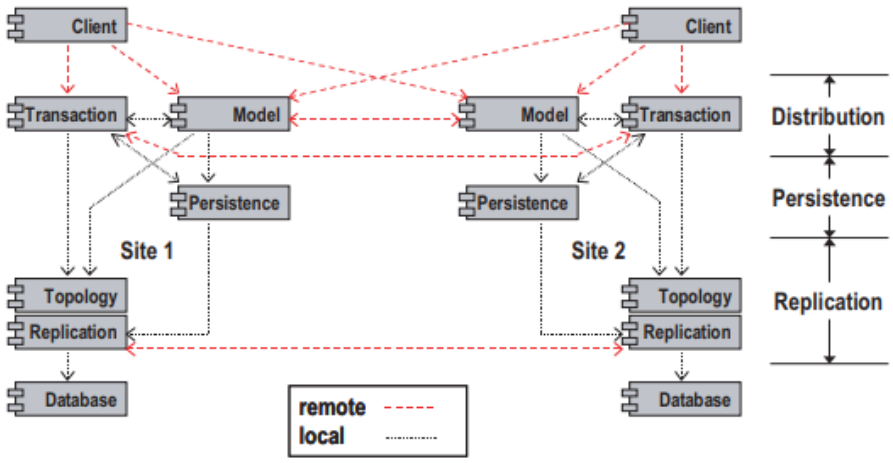
هر دو نویسنده به شدت درگیر طرح توسعه DTMS بودند:

Karl M. Goeschka دانشمند ارشد در Frequentis Nachrichtentechnik GmbH با دفتر مرکزی آن در اتریش و چندین شعبه در کشورهای دیگر (آمریکا، کانادا، آلمان) است. Frequentis رهبر بازار جهان برای ارتباطات صوتی بسیار سریع و دردسترس در کنترل ترافیک هوایی است. همچنین محصولات آن برای امنیت عمومی و ارتباطات امن استفاده می‌شود. با توجه به شخصیت خلاقانه توسعه‌ی DTMS او همچنین مدیر مسئول این پروژه نیز است.

Robert Smeikal دستیار تحقیق در دانشگاه وین در اتریش است. او در حال‌حاضر دوره‌ی دکترای خود را می‌گذارند و درگیر مشاوره در تیم توسعه Frequentis در مسائل عملی و مفهومی مهندسی نرم‌افزار است.

2. معماری DTMS و اجزا آن

شکل 2 معماری سیستم را با استفاده از نمودار اشیا (زبان مدل‌سازی UML) نشان می‌دهد.



شکل 2. معماری ارائه شده.

تصویر نشان می‌دهد که چگونه اجزا از یکدیگر به صورت محلی و یا از راه دور استفاده می‌کنند. یک سایت شامل اجزائ زیر است: Transaction، Model، Persistence، Topology، Replication و Database. جزء Client متعلق به سایت نیست. مشتری می‌تواند متصل به یک سایت دلخواه باشد. لطفا توجه داشته باشید که ارتباطات از راه دور بین اجزای از نوع یکسان برقرار است (به عنوان مثال. Replication محلی و Replication از راه دور). همانطور که در سمت راست نشان داده می‌شود، اجزاء Model و Transaction متعلق به جنبه‌ی توزیع هستند، جزء Persistence به جنبه تداوم و Topology و Replication به جنبه تکرار از معماری برمی‌گردد.

مدل. مولفه Model شامل مدل VCS است. این مدل

• پیچیده است، به عنوان مثال شامل بسیاری از کلاس‌ها، حالات کوچک شی و چند حالت شی در هر کلاس است.

• نیاز به اعتبارسنجی پیچیده دارد، به عنوان مثال بسیاری از محدودیت‌های الگوریتم که شامل بسیاری از حالات شی است.

• نیاز به عملیات پیچیده خواندن مربوط به بسیاری از اشیا، به‌عنوان مثال جمع‌آوری اطلاعات برای پیکربندی VCS.

هر مدل شی دارای یک سیستم گسترده ID منحصر به فرد است. هر کلاس دارای یک کارخانه در ارتباط است، که می‌تواند با نام کلاس (مانند سرویس نامگذاری) تخصیص یابد. کارخانه اشیاء را ایجاد و حذف می‌کند. کارخانه محدودیت اشیا را تایید می‌کند.

اشیاء و کارخانه‌ها در اجزاء Transaction ثبت‌نام می‌کنند. اشیاء حالت سریال شی خود را در Persistence ذخیره می‌کنند. توپولوژی جستجوهای کارخانه برای سایت واکنش‌گرا فعلی و مقداردهی اولیه بر این اساس بااستفاده از مدل راه دور و یا تداوم محلی است. کارخانه‌ها منابع شی سازگار (IORها، اشاره بهCORBA) ارائه می‌دهند که اشاره به شی محلی و یا از راه دور مرتبط با ID یا پرس‌وجو دیگر دارد. کارخانه‌ها IORها را در اشیاء از راه دور و با برقراری ارتباط از راه دور با کارخانه به دست می‌آورند. مشتریان می‌توانند IORها را برای هر شیء در کارخانه به دست آوردند.

رفتار تراکنشی و پشتکار برای همه مدل‌های اشیاء الزامی است. بنابراین، تمام مدل‌های اشیاء متدهای ()validate، ()commit، ()rollback و ()store را پیاده‌سازی می‌کنند.

مشتری. جزء سرویس‌گیرنده فراخوانی در اشیاء را به‌صورت مدل محلی و یا از راه دور انجام می‌دهد، که در آن موقعیت فیزیکی مدل شئ برای مشتری شفاف است. تعداد دلخواهی از اجزای مشتری می‌توانند به صورت همزمان در مدل کار کنند. جزء مشتری و اتمام تراکنش‌ها در Transaction شروع می‌شود.

توپولوژی. جزء Topology یک لیست از همه سایت‌ها و در دسترس بودن آنها و اطلاعاتی در مورد سایت حاضر برای یک شی خاص است. که توسط Model و Transaction مورد نیاز است. بنابراین، Topology به دیگر اجزاء می‌گوید که در کدام سایت‌ها باید دنبال یک شی باشد.

تراکنش. جزء Transaction، مسئول تراکنش همزمان سریال است. ()commit، ()rollback و ()validate را بر روی اشیاء در Model متعادل می‌کند، همچنین تراکنش را در persistence و مختصات تراکنش توزیع‌شده را از راه دور کنترل می‌کند (فاز 2 commit[3]).

در حال حاضر، کوچکترین واحد دانه دانه‌شدگی کل یک سایت است. از آنجا که انتظار نمی رود نوشتن تراکنش‌ها طول بکشد اعتبارسنجی شامل بسیاری از حالات شی از کلاس‌های مختلف است، که کافی است. ما بین تراکنش نوشتن و خواندن تمایز قایل می‌شویم.

ماندگاری. اجزاء Persistence حالات شی را با استفاده از جزء Replication بازیابی می‌کند. علاوه‌براین، روشی برای دسترسی به حالات شی در راه تراکنش‌ها، که از Transactionاستفاده می‌کنند فراهم می‌کند. Persistence

بین حالات شی که به خانه در سایت‌های محلی اختصاص داده می‌شود و حالات شی که به خانه در یک سایت از راه دور اطلاق می‌شود تفاوتی قائل نیست، اما همیشه به ادامه و بازیابی اجزاء Persistence محلی اصرار دارد. کارخانه‌ها همیشه Persistence محلی خود را برای بازیابی حالات شی همچنان ادامه می‌دهند. یک شی همچنان به ارائه شناسه خود ادامه می‌دهد.

پایگاه داده. جزء پایگاه‌داده اساسا برای ذخیره‌سازی ثابت کپسوله شده است و توابع اولیه‌ای برای ادامه و بازیابی داده‌ها فراهم می‌کند. همچنین ممکن است پایگاه داده محلی را در محفظه‌ای برای دسترسی بالا نگه دارد.

تکرار. جزء Replication در قلب معماری قرار دارد و الگوریتم تکرار را پیاده‌سازی می‌کند. که پروتکل تکرار را براساس Replication سایت‌های دیگر برای انتشار حالات شی، جهت ساخت حالات شی و محاسبه آمادگی شی انجام می‌دهد. و از جزء Database برای ادامه اطلاعات به صورت محلی استفاده می‌کند و روشی برای دسترسی به حالت شی ذخیره شده فراهم می‌کند.

3. مطالب فراگرفته شده

ما مهم‌ترین مطالب را با تمرکز بر روی جنبه‌های مفهومی به جای جزئیات پیاده‌سازی و یا یافته‌ها در مورد محصولات COTS و ابزار ارائه می‌دهیم.

معناشناسی روش‌ها در مقابل خواندن/نوشتن: مقدار قابل توجهی نظریه در مورد موضوع روش بهره‌برداری معناشناسی به‌منظور افزایش در دسترس بودن برخی از روش‌ها وجود دارد (به‌عنوان مثال [8]، که در آن "روابط تقاطع" جریان اطلاعات را میان توابع فراخوانی می‌کند). در عمل، ما قادر به استفاده از هر روش طبقه‌بندی نسبت به روش خواندن و روش نوشتن، به ویژه با توجه به موارد استفاده مشترک نیستیم. به غیر از افزایش قابل توجه پیچیدگی اجرا (یک روش تکرار در نظر بگیرید، که در مورد روش معناشناسی هر نوع شی و محاسبه در دسترس بودن اشیاء می‌داند) سیستمی غیر قابل درک از نقطه نظر کاربران ارائه می‌دهد.

نگاشت هویت شی به مرجع شی: به‌دست آوردن مرجع شی (به‌عنوان مثال محل یک شی) با توجه به شناسه شیء (به‌عنوان مثال نام شی) همیشه از نگرانی برخوردار است اگر اشیاء خارج از فرآیند یا حتی از راه دور باشند (در مقایسه با سرویس نامگذاری CORBA [1] و یا نامگذاری جاوا و رابط راهنما [2]). بنابراین چنین سرویس نامگذاری نیاز به تداوم اساسی دارد، اگر اشیاء در مکان‌های مختلف با توجه به سناریوی تخریب خاص زندگی کنند این موضوع پیچیده می‌شود. از یک طرف، اطلاعات سرویس نامگذاری در مورد ارجاع اشیاء حفظ می‌شود (که در آن یک مرجع محل اسیا را تعیین می‌کند)، از سوی دیگر، جزء تکرار تصمیم می‌گیرد که اگر تخریب رخ دهد اشیاء نمونه کجا قرار گیرند. بدین ترتیب، نگاشت شناسه به مرجع بسیار وابسته به مکانیسم‌های زیر بنایی برای ارائه تحمل‌پذیری خطا است. علاوه‌براین، سرویس نامگذاری با خرابی امن باید در هر سایت و در هر زمان دردسترس باشد، چراکه سرویس نامگذاری باید خودش تحمل‌پذیری خطای بالایی داشته باشد.

در این مطالعه، یک شناسه شی با استفاده از دو مرحله ارائه می‌کنیم. در ابتدا، جزء توپولوژی برای سرویس نامگذاری حاضر پرس‌وجو می‌کند (کارخانه‌ها در DTMs)، که می‌تواند بسته به سناریوی تخریب به صورت از راه دور یا محلی باشد. در مرحله دوم، در نهایت شناسه شی حل و فصل می‌شود.

علاوه بر این، استفاده اجباری از شناسه شی برای حفظ مراجع (ما آنها را "مراجع نرم") مشکلی را معرفی می‌کند که نیاز به حل هر شناسه در هر زمانی (که معمولا بازده بدی دارد) و یا حفظ یک نوع شناسه/ مرجع در cache و در درون هر شی دارد.

دانه دانه شدگی: تعریف واحدی از دانه دانه‌شدگی برای عملیات سیستم یک تصمیم طراحی اساسی است. که می‌تواند برای بخش‌های مختلف سیستم که متضمن پیاده‌سازی پیچیدگی و رفتار سیستم است متفاوت باشد. مرزهای دانه دانه‌شدگی در محدوده مدل شی‌گرا محدود به تک تک ویژگی‌ها تا کل مدل است. اگرچه ما از دانه دانه‌شدگی‌های متفاوتی برای اجزاء متفاوت استفاده می‌کنیم:

تداوم: رسیدن به ماندگاری داده شی با استفاده از کل اشیاء به‌عنوان واحد دانه دانه شدگی، نسبتا آسان است، چرا که کدهای لازم را از مدل‌های اطلاعات با استفاده از رابط‌های برنامه کاربردی (نرم‌افزار برنامه‌نویسی رابط) در مطالعه ما (مهندسی نرم‌افزار با کمک کامپیوتر) تولید می‌کنیم.

تراکنش‌ها و همزمانی: در حال حاضر، یک سایت واحدی از دانه دانه‌شدگی است. از آنجا که این موضوع به وضوح کافی نیست، ما در تلاش برای استفاده از دانه دانه‌شدگی شی هستیم. این امر مستلزم توانایی الحاق هر شی با تراکنش خاص و محاسبه امکان commit است. علاوه‌براین، پیچیدگی بیشتری معرفی شده است چرا که عملیات نوشتن "غیرمستقیم" بنا به روابط شی و به‌دلیل قفل ضروری اشیاء "درگیر" است که در طول محدودیت اعتبارسنجی استفاده می‌شود. ما قصد داریم کد لازم را که به وضوح پیچیده است تولید کنیم.

معماری دو لایه: طراحی معماری دو لایه ما به‌طور عمده به یک نیاز متمرکز است: دسترسی خوانده شدن هر شی باید در همه زمان ممکن باشد، در حالی که دسترسی خوانده شدن باید در یک سایت خاص (سایت اصلی آن) ممکن باشد. برای رهایی اجرای شی از تکرار همزمان (نگه داشتن داده‌ها و هماهنگ کردن commit با استفاده از پروتکل کنترل مانند یک گروه ارتباطات [4])، تصمیم به استفاده از تکرار غیرهمزمان گرفتیم (commit اول، نگه داشتن پس از آن استفاده از صف). تکرار غیرهمزمان ثبات را به خطر می‌اندازد، اگر چند نمونه از یک شی در سیستم زندگی کند (به‌عنوان مثال اشیاء قبل از تغییر از یک سایت از راه دور قابل دسترسی شوند). بنابراین، در طول حیات سیستم سالم، ما از راه‌اندازی معمولی سیستم توزیع‌شده استفاده می‌کنیم: هر شی فقط یک بار نمونه‌برداری می‌شود، اشیاء به روشی همزمان با اشیاء دیگر ارتباط برقرار می‌کنند و مفاهیم محاسبات توزیع شده را می‌توان به کار برد. این باعث ایجاد لایه فوقانی، به نام لایه‌ی توزیع می‌شود. علاوه‌براین، ادامه شی داده به‌صورت غیرهمزمان برای آماده‌سازی سیستم سناریو تخریب منتشر می‌شود. این باعث ایجاد لایه‌های پایین‌تر، به نام لایه تکرار می‌شود. همه ملاحظات فوق به مدل اشیا، همراه با جزء "مدل" به کار می‌رود. در مقابل، اجزاء"سیستم" در هر سایت وجود دارد و مدل اشیاء توزیع شده را مدیریت می‌کند.

با اقتباس از تجربه ما، مفهوم مطرح شده در اینجا استفاده از تکرار غیرهمزمان داده شی به جای تکرار همزمان در سطح نمونه است.

CBSE: تداوم و تراکنش معمولا به‌عنوان سرویسی از اجرای مدل جزء (CMI) ارائه می‌شود، اما از آنجایی تعامد مورد نظر است (به عنوان مثال جایگزین الگوریتم تکرار) CMI باید براساس جزء باشد. DTMS با استفاده از یک مدل جزء خاص اجرا نمی‌شود اما طراحی آن براساس اصل اساسی CBSE است: انسجام قوی نگرانی و استانداردهای رابط همراه. بنا به تجربه ما، این رویکرد، تلاش در آغاز پروژه را افزایش می‌دهد، اما بعد از آن هم از نظر مسائل فنی و هم سازمانی کاهش می‌یابد. تا کنون، نرم‌افزار ما در سه طرح برای مشتریان مختلف و بر روی سیستم‌عامل‌های مختلف (ویندوز مایکروسافت و سان سولاریس) استفاده شده است و تنها دو جزء به جای نیاز مشتری تغییر کرده است: بقیه چارچوب بدون تغییر باقی مانده و پایدار است.

با این حال، CBSE با یک نقطه ضعف مواجه است. با توجه به کپسوله‌سازی اجزاء، نگاشت‌های متعددی از رابط اجزاء-داخلی به اجزاء خارجی ("ارائه شده") به‌جای استفاده‌ی (شی به شی) لازم است. که بازده زمان اجرا را کاهش می‌دهد و می‌تواند بطور قابل توجهی به اندازه سیستم رشد کند. از این رو، یک رابطه پنهان بین انسجام اجزاء و عملکرد زمان اجرا وجود دارد.

پهنای باند-WA : در مرحله طراحی پروژه، ما بین تماس‌های در حال انجام، خارج از روند و CORBA تمایزی قائل نشدیم، چرا که انتظار می‌رود میان‌افزار هر گونه مسائل مرتبط را از نقطه نظر برنامه‌نویسان نرم‌افزار مخفی کند. اگر چه این درست است، ما پس از آن تجربه کردیم که چنین استفاده‌ی غیر قابل تنظیمی از تماس‌های CORBA 2مگابیت از لینک‌های ما را به طور چشمگیری اشغال می‌کند. به غیر از بهینه‌سازی CORBA، ما مجبور به دقت بیشتر شدیم که در آن تماس‌های CORBA واقعا لازم است.

حالات شی برای نگاشت به پایگاه داده: ما پایگاه‌داده‌ی خود را به جای حالات اشیا با استفاده از یک جدول با سه ستون طراحی کردیم: شناسه شی، نوع شی و شی داده با فرمت XML. این راه‌اندازی، انعطاف‌پذیری بالایی در رسیدگی

به تغییرات در مدل بدون تغییر یا طرح پایگاه داده و یا تداوم و جزء تکرار فراهم می‌کند. با این حال، ما با برخی اشکالات مواجه شدیم:

• از آنجا که هیچ اعتبارسنجی ایستایی نمی‌تواند استفاده شود، پایگاه داده تناقض بین حالات شی را اجازه می‌دهد. هر چند واضح است، اما این موضوع ثابت شده است، چرا که اکنون حتی شکست‌های کوچک در مدل شی ممکن است پایگاه داده‌ای در یک وضعیت نامناسب ارائه دهد.

• پرس‌وجوهایی که به‌طو مستقیم در پایگاه داده کارعمل می‌کنند برای پیاده‌سازی بسیار پرزحمت هستنتد، از آنجا که نه تنها SQL بلکه XPath / XQuery را می‌توان مورد استفاده قرار داد.

• XML برای یک مشتری انسان قابل خواندن است، اما بنا به اطلاعات زائد بی ارزش است. اندازه داده‌های XML به سرعت و با رشد اندازه سیستم در حال افزایش است و در نتیجه کارایی آن به شدت کاهش می‌یابد. خوانایی انسان از این ساختارهای بزرگ و پیچیده تقریبا در حد صفر است. بنابراین، در حال حاضر یک فرمت باینری برای جایگزینی XML در سطح پایگاه داده توسعه می‌دهیم.

ساختار سلسله‌مراتبی: سلسله‌مراتب ثابت کرده است که یکی از ساختارهای قوی، امکان‌پذیر و عملی است [9]. از سوی دیگر، ساختار مدل نمی‌تواند محدود شود، زیرا روابط چند به چند بسیاری مورد نیاز است. بنابراین، ساخت یک ساختار سلسله‌مراتبی با استفاده از تجمع منابع مفید است. در این مقاله یک مدل معرفی می‌شود که دقیقا خواستار هر شی قابل دسترسی با یک تجمع سلسله‌مراتبی است.

موتور پرس‌وجو: دو نوع پرس‌وجو ساده پشتیبانی می‌شود: دریافت شی با ID و دریافت تمام اشیاء از یک نوع خاص. برای اجرای کارآمد هدف UI، پرس‌وجوهای اضافی برای انتقال از طریق سلسله مراتب شی ضروری است، به‌عنوان مثال، دریافت جد یا فرزندان یک شی که در آن شرایط اضافی در صفات برآورده شده است. گرچه زبان پرس‌وجو توصیفی مانند OQL (زبان جستجو شی [1]) مطلوب است، پرس‌وجوهای ذکر شده کافی به نظر می‌رسد.

انتشار عملیات حذف: هنگامی که یک شی در موضع حذف قرار می‌گیرد، ممکن است اشیاء دیگری وجود داشته باشند که به این شیء اشاره دارند. مشابه پایگاه داده‌های رابطه‌ای، تعیین می‌شود که آیا مرجع و شی رجوع شده حذف

می‌شود. این ممکن است در نتیجه عملیات حذف آبشاری و یا در محدودیت حذف هنگام محدودیت نقض اتفاق بیافتد. گاهی اوقات این نتایج کاملا در عملیات پیچیده‌ی حذف در سمت سرویس‌گیرنده پیاده‌سازی می‌شود. ما هنوز یک راه‌حل کلی مناسب نداریم، اما یک مسئله مهم برای بررسی بیشتر است.

4. کارهای آینده

بهبودهای زیر مشخص شده است:

• پشتیبانی از یک موتور پرس‌وجو، که اجازه‌ی پرس‌وجوهای توصیفی با استفاده از OQL را می‌دهد.

• پشتیبانی از دانه‌دانه‌شدگی قفل برای رسیدن به عملکرد همزمانی بهتر در دسترسی مدل برای مشتریان.

• استفاده از روش‌های نوشتن در مدل اشیاء در همه سایت‌ها حتی در طول تخریب سیستم، از نیازمندی‌های بزرگ آینده است.

• بهره‌برداری بهتر از انواع تراکنش‌ها (در حال حاضر خواندن / نوشتن دردسترس) در رابطه با تجارت بین دردسترس بودن اشیا و نیازمدی ثبات مورد نظر است [5، 10].

با بهبود نقاط ضعف سیستم فعلی، ما قادر به ارائه دسترس بالا و توزیع پلت‌فرم برای سیستم‌های شی هستیم.

References

[1] http://www.omg.org/. The Object Management Group.
[2] http://www.sun.com/. Sun Microsystems.
[3] P. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. AddisonWesley, 1987.
[4] K. Birman. The process group approach to reliable distributed computing. Communication of ACM, 36(12):37–53, December 1993.
[5] H. Garcia-Molina and G. Wiederhold. Read-only transactions in a distributed database. ACM Transactions on Database Systems, 7(2):209–234, June 1982.
[6] G. Heineman and W. Councill. Component-Based Software Engineering. Addison-Wesley, 2001.
[7] A. Helal, A. Heddaya, and B. Bhargava. Replication Techniques in Distributed Systems. Kluwer Academic Publishers, 1995.
[8] M. Herlihy. A quorum consensus replication method for abstract data types. ACM Transactions on Computer Systems, 4(1):32–53, February 1986.
[9] P. Kahn. Information architecture: a new discipline for organizing hypertext. In Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, pages 1–2. ACM, September 2001.
[10] D. Skeen. Achieving high availability in partitioned database systems. In Proceedings of the International Conference on Data Engineering, pages 159–166. IEEE, 1985.