تبدیل مدل دوسویه بر اساس چارچوب اپسیلون

لیلا صمیمی دهکردی
دانشکده مهندسی کامپیوتر- دانشگاه اصفهان - اصفهان - ایران
پست الکترونیکی: samimi@eng.ui.ac.ir

بهمن زمانی
دانشکده مهندسی کامپیوتر- دانشگاه اصفهان - اصفهان - ایران
پست الکترونیکی: zamani@eng.ui.ac.ir

شکوفه کلاهدوز رحیمی
دانشکده مهندسی کامپیوتر- دانشگاه اصفهان - اصفهان - ایران
پست الکترونیکی: shrahimi@eng.ui.ac.ir

چکیده: تولید مدل‌گرا[1] از ایدهٔ ارتقای سطح تجرید به منظور خودکارسازی تولید کد استفاده می‌کند. در این روش تولید، از روی مدل و با به‌کارگیری مجموعه روش‌هایی تحت عنوان تبدیل مدل، کد پیاده‌سازی به صورت (نیمه) خودکار تولید می‌شود. کلیه فعالیت‌های ممکن در این روش با استفاده از تبدیل‌ها انجام می‌گیرد، به همین دلیل تبدیل را قلب و روح تولید مدل‌گرا می‌گویند. یکی از انواع فعالیت‌ها تبدیل مدل به مدل است که در ساده‌ترین حالت، یک ارتباط تک‌سویه بین مدل مبدأ و مقصد تعریف می‌شود که فقط مدل مقصد از روی مدل مبدأ قابل تولید است. در حالت کلی‌تر، مدل‌های مبدأ و مقصد مستقلاً تغییر می‌یابند و در نتیجه، نیاز به تبدیل دوسویه مطرح می‌شود. روش‌های متعددی برای تعریف و اجرای یک تبدیل دوسویه در سال‌های اخیر ارائه شده است. در این مقاله، با توجه به چالش‌های این روش‌ها، روشی جدید بر مبنای چارچوب مدل رانده‌ی اپسیلون و تکنیک‌های ردیابی‌پذیری برای تبدیل مدل دوسویه ارائه می‌شود. برای نمایش قابلیت‌ها و چگونگی استفاده از روش پیشنهادی، آن را روی یک محک شناخته شده در زمینه تبدیل دوسویه اجرا می‌کنیم. از جمله قابلیت‌های روش پیشنهادی پشتیبانی از صحت، جامعیت و انتشار تغییر است.

واژه‌های کلیدی: تولید مدل‌گرا، تبدیل مدل دوسویه، ردیابی‌پذیری، چارچوب اپسیلون

1. مقدمه

امروزه روش‌های تولید نرم‌افزار به دنبال ارتقای سطح تجرید در جهت کاهش پیچیدگی‌های برنامه‌ها هستند. در واقع، هدف از ارتقای سطح تجرید برنامه‌ها این است که برنامه‌نویس برای ماشین تعیین کند که **چه کاری** باید انجام دهد، نه این که **چگونه** آن را انجام دهد [1]. یکی از جدیدترین روش‌هایی که از ایده ارتقای سطح تجرید به منظور خودکارسازی تولید کد استفاده می‌کند، تولید مدل‌گرا است. تولید مدل‌گرا، یک روش تولید است که از مدل به عنوان محصول اولیه در فرآیند تولید استفاده می‌کند؛ به طوری که در آن، از روی مدل و با به‌کارگیری مجموعه روش‌هایی تحت عنوان تبدیل مدل، کد پیاده‌سازی به صورت (نیمه) خودکار تولید می‌شود. با توجه به آن که برنامه را می‌توان ترکیبی از ساختمان داده‌ها و الگوریتم در نظر گرفت، نرم‌افزار را نیز می‌توان ترکیبی از مدل‌ها و تبدیل‌ها دانست [2]. با در نظر گرفتن کد و مدل به عنوان مصنوعات قابل تولید، چهار نوع تبدیل به صورت مدل به مدل، مدل به کد، کد به مدل و کد به کد بررسی می‌شود [3]. تمرکز این مقاله روی تبدیل مدل به مدل است.

---

[1]Model-Driven Development (MDD)

تبدیل با هدف خودکارسازی فعالیت‌هایی از قبیل پالایش[2]، بهسازی[3]، تغییر در سطح تجرید مدل‌ها، بازبینی و تصحیح آن‌ها، مهندسی رو به جلو[4]، تولید کد[5] و مهندسی معکوس[6] (به منظور تولید مدل سیستم از کد اجرایی) انجام می‌شود؛ به همین دلیل، تبدیل را قلب و روح تولید مدل‌گرا می‌گویند [4]. در ساده‌ترین حالت، تبدیل، برای یک ارتباط تک‌سویه[7] بین مدل مبدأ و مقصد به‌کار گرفته می‌شود. در حالت کلی‌تر، مدل‌های مبدأ و مقصد مستقلاً تغییر می‌یابند [5]. در این حالت، چارچوب مدل باید با استفاده از تبدیل مدل دوسویه غنی شود، تا بتواند سازگاری[8] میان مدل مبدأ و مقصد را حفظ کند.

در تولید مدل‌گرا، تبدیل مدل دوسویه دو وظیفه دارد [6]: 1) بررسی سازگاری: باید بتواند تعیین کند که چه زمانی دو مدل بر طبق تعریف تبدیل سازگار هستند؛ 2) تحمیل[9] (یا بازیابی) سازگاری: باید بتواند یک جفت مدل ناسازگار را گرفته و یکی از مدل‌ها (از قبل تعیین شده) را تغییر دهد تا سازگاری برقرار شود.

روش‌های متعددی برای تعریف و اجرای یک تبدیل دوسویه در سال‌های اخیر ارائه شده است. اکثر این روش‌ها از نظر سبک زبان اعلانی[10] هستند و همین باعث می‌شود که اجرای تبدیل در آن‌ها با ابهاماتی همراه باشد [7]. همچنین زبان‌های اعلانی دارای نشانه-گذاری[11] خاص و پیچیده‌ای هستند که تولیدکننده مجبور است آن را فراگیرد [8]. از طرف دیگر، زبان‌های دستوری یا روش‌های تک‌سویه نیز نمی‌توانند اکثر نیازمندی‌های لازم دوسویگی را فراهم کنند. در [9]، ما روشی را بر مبنای زبان اعتبارسنجی اپسیلون[12] و تکنیک‌های ردیابی‌پذیری[13] به نام EVL+trace مطرح کردیم که به حل مشکلات روش‌های دوسویه دیگر همانند ابهامات ذاتی زبان-های دوسویه پرداخته و ویژگی‌های لازم دوسویگی را سهولت می‌بخشد. در این مقاله، نحوه چگونگی کار با روش EVL+trace را روی یک محک[14] شناخته شده [10] نشان می‌دهیم. با اجرای این روش روی نمونه‌های آزمون مربوط به این محک مشاهده می‌شود که روش پیشنهادی به خوبی ویژگی‌های دوسویگی را ارضا می‌کند.

در ادامه مقاله، ابتدا مفاهیم اولیه تولید مدل‌گرا از قبیل مهندسی مدل گرا، محصولات اولیه مدل، تبدیل مدل و انواع آن در بخش 2 بیان می‌شود. در بخش 3 کارهای تحقیقاتی انجام شده در زمینه تبدیل مدل دوسویه بررسی می‌شوند. سپس، در بخش 4 چارچوب اپسیلون و زبان اعتبارسنجی آن که در این مقاله استفاده شده معرفی می‌شوند. روش پیشنهادی EVL+trace، محک مورد بررسی، چگونگی پیاده‌سازی روش پیشنهادی روی این محک و نتایج اجرا در بخش 5 ارائه خواهد شد. بخش 6 به جمع‌بندی مقاله می‌پردازد.

2. مفاهیم اولیه

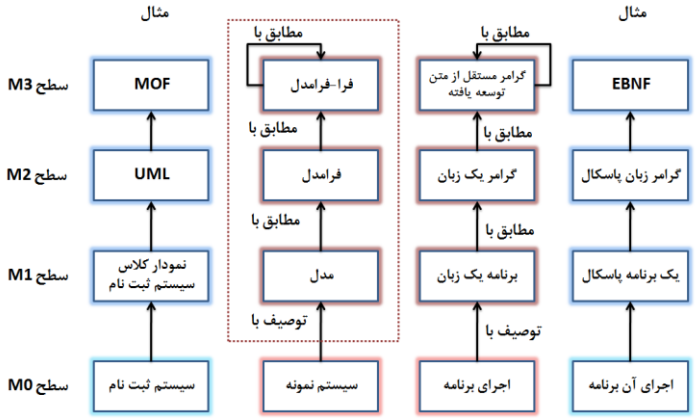
سابقه ارتقای سطح تجرید به معرفی زبان اسمبلی به عنوان تجریدی روی کد ماشین بازمی‌گردد؛ پس از آن، زبان‌های نسل سوم مطرح شدند که در آن برخی از وظایف سطح پایین ماشین را به جای برنامه‌نویس به برنامهٔ مترجم (کامپایلر) محول می‌کردند [11]. زبان‌های شیءگرا، تجریدهای بالاتری مانند نوع داده تجریدی را مطرح کردند. تولید مدل‌گرا این سنت را ادامه داد و با معرفی تجرید بالاتر از کد، یعنی مدل، آن را در سطوح مختلف چرخهٔ حیات نرم‌افزار گسترش داد. به طور خلاصه، هر جهش در تولید نرم‌افزار، با ارتقای سطح تجرید و در نتیجه، مستقل شدن از بُن‌سازه[15] منجر به نرم‌افزارهایی با قابلیت حمل بالا، افزایش کیفیت، بهره‌وری و همکنش‌پذیری[16] و در عین حال، کاهش زمان عرضه محصول نرم‌افزاری به بازار می‌شود [11].

---

2 Refinement
3 Refactoring
4 Forward Engineering
5 Code generation
6 Backward Engineering
7 Unidirectional
8 Consistency
9 Enforcement
10 Declarative
11 Notation
12 Epsilon Validation Language (EVL)
13 Traceability
14 Benchmark
15 Platform
16 Interoperability

این روش تولید زیرمجموعه‌ای از **مهندسی مدل‌گرا** است؛ مهندسی مدل‌گرا روش نویدبخشی را برای حل ناتوانی زبان‌های نسل سوم در جهت کاهش پیچیدگی بـن‌سـا ز ۀ نرم‌افزار و بیان مؤثر مفاهیم دامنه پیشنهاد می‌دهد [12]. بنابراین، مهندسی مـدل‌گرا تکامـل یافته ابزارهای «مهندسی نرم‌افزار به کمک کامپیوتر[۱۷]» می‌باشد و نوع خاصی از **مهندسی مدل-مبنا**[۱۸] محسوب می‌شـود. مهندسـی مدل-مبنا فرآیندی است که مدل‌های نرم‌افزاری در آن نقش ویژه‌ای دارند، اگرچه ممکن است مدل، محور اصلی در تولیـد آن نباشـد [2]. در دهه 80، فناوری شیءگرا با اصل «هر چیز یک شیء است» بیان می‌شد، اما در مهندسی مدل‌گرا اصل مهـم بـا عبـارت «هـر چیز یک مدل است» بیان می‌شود [13]. می‌توان **مدل** را به عنوان انتزاعی از یک سیستم واقعی که بر طبق هـدف خاصـی بـه وجـود آمده، تعریف کرد. به همان صورتی که برنامه‌های یک زبان برنامه‌نویسی بر اساس دسـتور زبـان آن نوشـته می‌شـوند، مـدل نیـز بایـد منطبق با زبان مدل که در اصطلاح به آن فرامدل[۱۹] می‌گوییم، باشد.

زبان مدل‌سازی یکپارچه[۲۰] می‌تواند به عنوان ایده‌ای برای تعریف فرامدل استفاده شود، البته نمی‌توانـد بـه تنهـایی در همـه زمینـه‌ها مفید باشد. بنابراین، برای جلوگیری از تنوع فرامدل‌های ناسازگار نیاز به زبانی احساس می‌شد که با آن بتوان همه فرامدل‌ها را تعریـف کرد، که در اصطلاح به آن **فرا-فرامدل**[۲۱] گفته می‌شود [2]. گروه مدیریت شیء(OMG)[۲۲] برای زبان فرا-فرامدل، استاندارد تسهیلات فرا شیء[۲۳] را معرفی کرد. در شـکل 1، معمـاری چهـار لایـه‌ای فرامدل‌سـازی و تنـاظر ایـن لایـه‌ها بـا مفـاهیم پایـه‌ای در زبان‌هـای برنامه‌سازی نشان داده شده است. هر لایه بر طبق لایه بالاترش تعریف می‌شود و با رابطه تطابق به آن متصل می‌گردد. فقط بـالاترین لایه به صورت بازگشتی خود را تعریف می‌کند. هر فرامدل در لایۀ دو، خود زبانی برای تعریف دامنه‌ای خاص خواهد بود. بـرای مثـال، UML زبانی برای توصیف مدل‌های نرم‌افزاری شیءگرا می‌باشد.



شکل (1): معماری چهار لایه‌ای تولید مدل‌گرا برگرفته از [13]

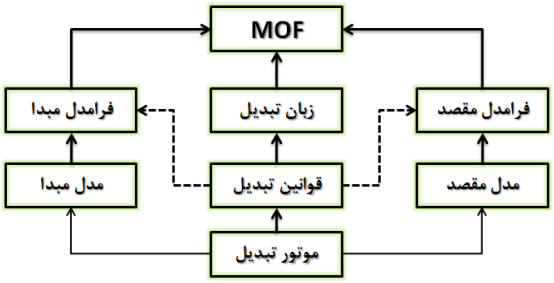
2-1 تبدیل مدل

طبق دیدگاه تولید مدل‌گرا، مدل‌ها و فرامدل‌ها، شهروندان درجه یک هر فرآیند تولید محسوب می‌شـوند و سـازوکاری کـه مـدل‌های متفاوت را به هم متصل می‌کند، با عنوان تبدیل مدل شناخته می‌شود [14]. ابزارهای فرامدل‌سازی سه روش مختلف را بـرای تعریـف تبدیل، به عنوان یک محصول نرم‌افزاری ممکن می‌سازند، که عبارتند از [4]: 1- دستکاری مستقیم مدل[۲۴]، 2- نمایش میـانی[۲۵] و 3- **زبان تبدیل**[۲۶]. روش سوم به دلیل ارتقای سطح تجرید، قوی‌ترین نوع توصیف است. زبان تبدیل، یک زبان خاص دامنـه[۲۷] اسـت کـه

---

[17] Computer-Aided Software Engineering (CASE)
[18] Model-based Engineering (MBE)
[19] Metamodel
[20] Unified Modeling Language (UML)
[21] Meta-metamodel
[22] Object Management Group
[23] Meta Object Facility (MOF) : http://www.omg.org/mof
[24] Direct Model Manipulation
[25] Intermediate Representation
[26] Transformation Language
[27] Domain Specific Language (DSL)

سازوکارهایی را برای تعریف، ترکیب و اجرای تبدیل فراهم می‌کند [4]. با دنبال کردن اصل «هر چیزی یک مدل است»، تبدیل مدل نیز می‌تواند همانند یک مدل دیده شود (شکل 2).



شکل (2): تبدیل مدل در مهندسی مدل رانده برگرفته از [13]

همان طور که مدل‌های مبدأ و مقصد با فرامدل متناظرشان تطبیق دارند، قوانین تبدیل نیز با زبان تبدیل مطابقت دارند و همه فرامدل‌ها، یعنی فرامدل مبدأ، فرامدل مقصد و زبان (فرامدل) تبدیل، با فرا-فرامدلی، یعنی لایهٔ سه از معماری فرامدل‌سازی در شکل 1، تطبیق دارند.

2-2 تبدیل مدل دوسویه

زبان‌های تبدیل از نظر جهت اجرا به دو دستهٔ زبان تک سویه و زبان دوسویه تقسیم می‌شوند. **تبدیل دوسویه**[۲۸] سازوکاری برای نگهداری سازگاری میان دو (یا بیشتر) منبع اطلاعاتی مرتبط است [15]. تا کنون، کاربردهای تبدیل دوسویه در حوزه زبان‌های برنامه‌نویسی، تبدیل‌های گراف، مهندسی نرم‌افزار و پایگاه‌داده‌ها شناخته شده است [15]. پذیرش مفهوم دوسویگی در مهندسی مدل‌گرا در سال 2005 توسط OMG مطرح شد که شامل یک زبان دوسویه در پروژه تبدیل استاندارد QVT می‌شد [16]. زبان‌های تبدیل دوسویه روش رسمی برای تبدیل است که در آن هر برنامه، تبدیل رو به جلو[۲۹] و تبدیل رو به عقب[۳۰] را همزمان توصیف می‌کند. مزیت تبدیل‌های دوسویه، تضمین برقراری سازگاری توسط ساختار زبان است [15]. در حال حاضر دو روش برای تبدیل مدل دوسویه وجود دارد [6]: 1) با نوشتن دو برنامه به یک زبان تبدیل تک‌سویه برای انجام تبدیل‌های رو به جلو و معکوس، 2) با نوشتن یک برنامه به یک زبان تبدیل مدل دوسویه.

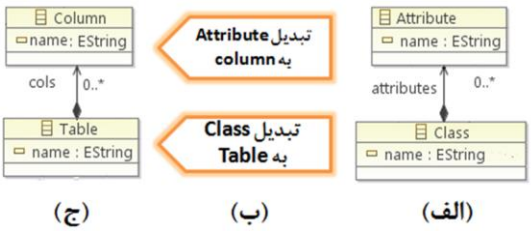
حتی اگر فرآیند بازیابی سازگاری بین دو مدل، به صورت دستی انجام شود، بررسی خودکار این که مدل‌ها سازگار هستند یا خیر می‌تواند ارزشمند باشد [16]؛ گاهی، بازیابی سازگاری به دلیل محدودیت در زمان اجرا، برقرار نشدن تمام شرایط و یا وجود چندین جواب (چند مدل مقصد)، به طور کامل ممکن نیست، در این موارد تبدیل به صورت جزئی انجام می‌گیرد [17]. اهمیت دوسویگی در عمل به‌واسطه سناریوهای کاربردی آشکار شده است؛ از جمله کاربردهای دوسویگی می‌توان به انتشار تغییر[۳۱] ، همگام‌سازی[۳۲]، مدل‌سازی چندگانه[۳۳] ، تکامل نرم‌افزار[۳۴] و مهندسی رفت و برگشت[۳۵] اشاره کرد [15].

2-3 مثالی ساده از تبدیل مدل

فرض کنید، بخواهیم یک مدل از فرامدل شیءگرا را به مدلی از فرامدل رابطه‌ای تبدیل کنیم. فرامدل ساده شدهٔ شیءگرا در شکل 3-الف، تبدیل موردنظر در شکل 3-ب و فرامدل ساده شده‌ی رابطه‌ای در شکل 3-پ نشان داده شده است. در ساده‌ترین حالت، برنامه تبدیل، با استفاده از دو قانون تبدیل رده به جدول و تبدیل صفت به ستون تعریف می‌شود. در صورتی‌که مدل مبدأ شامل یک رده به نام Student و سه صفت به نام‌های firstName، lastName و ID باشد، طبق تعریف تبدیل رو به جلو (از مبدأ به مقصد)، مدل مقصدی که حاصل می‌شود، باید شامل یک جدول با نام Student با سه ستون، هم‌نام با صفات باشد که در شکل 4 قابل مشاهده است.

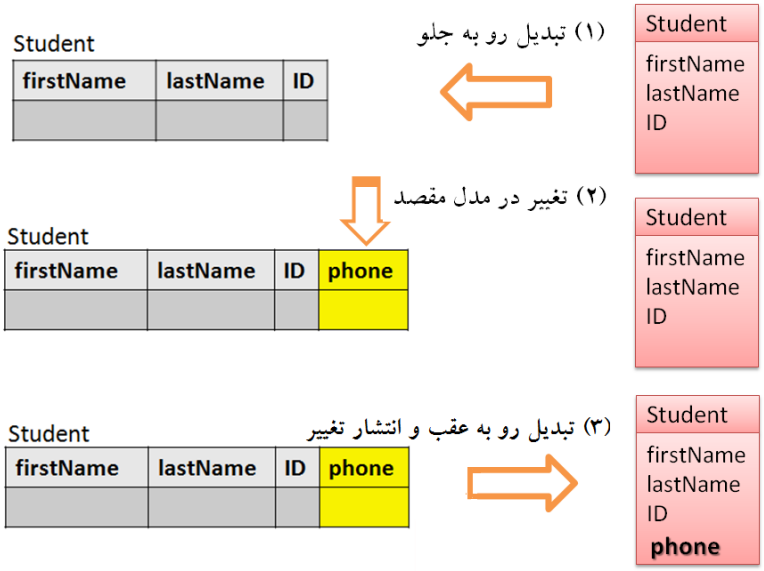
---

[28] Bidirectional Transformation (Bx)
[29] Forward transformation
[30] Backward transformation
[31] Change propagation
[32] Synchronization
[33] Multiview Modeling
[34] Software Evolution
[35] Round-trip Engineering



شکل (3): الف) فرامدل شیءگرا، ب) قوانین تبدیل و ج) فرامدل رابطه‌ای

فرض کنیم پس از اجرای تبدیل، کاربر بخواهد تغییری در مدل مقصد بدهد. برای مثال، همانند شکل 4- مرحله (2) ستونی به نام phone را به جدول اضافه کند. در این حالت، سازگاری بین مدل مبدأ و مقصد از بین می‌رود، زیرا لازم است به ازای هر صفت در رده، یک ستون در جدول وجود داشته باشد و بالعکس. اما پس از تغییر کاربر در مدل مقصد، هیچ صفتی به نام phone در مدل مبدأ نخواهد بود.



شکل (4): مثالی ساده از تبدیل دوسویه

برای بازیابی سازگاری بین مدل‌ها باید تبدیل را در جهت معکوس از مقصد به مبدأ (تبدیل رو به عقب) انجام دهیم تا تغییر ایجاد شده در مدل مقصد به مبدأ انتشار یابد. این عمل در مرحله (3) از شکل 4 نشان داده شده است. باید در نظر داشت که گاهی همزمان و به صورت مستقل، مدل مبدأ و مقصد تغییر می‌یابند؛ در این‌حالت، علاوه بر آن‌که تغییرات از مقصد به مبدأ (یا بالعکس) انتشار می‌یابد، باید تغییرات مدل مبدأ نیز حفظ شود. به این عمل همگام‌سازی بین مدلی گفته می‌شود.

3. کارهای تحقیقاتی در زمینهٔ تبدیل‌های دوسویه

دراین بخش، روش‌های متعدد تبدیل مدل دوسویه را بررسی می‌کنیم. در اکثر روش‌های دوسویه، زبان‌های تبدیل مدل دوسویه بررسی می‌شوند؛ هر کدام از این زبان‌ها دارای مزایا و معایبی هستند که در ادامه به آن خواهیم پرداخت. در سایر روش‌ها، زبان‌ها یا روش‌های تک‌سویه برای نوشتن تبدیل دوسویه به نوعی تغییر می‌یابند.

زبان *QVT-R* یک زبان رابطه‌ای تعریف شده توسط OMG است و بر اساس MOF می‌باشد [18]. نحو این زبان می‌تواند متنی یا گرافیکی باشد و از تبدیل دوسویه به صورت اعلانی پشتیبانی می‌کند. تبدیلات تعریف شده اهداف مختلفی دارند: 1) برای بررسی سازگاری دو مدل 2) برای تحمیل سازگاری با تغییر در مدل مقصد 3) برای همگام‌سازی دو مدل 4) برای انجام تبدیل‌های درجا[۳۶] این زبان هنوز دارای ابهامات زیادی است [6]. در [19] سعی شده است که برخی از ویژگی‌های معنایی این زبان برای غلبه بر

[36] In-place

ابهامات موجود بررسی شود. یکی از ابزارهایی که این زبان را پیاده‌سازی می‌کند،medini [37] نام دارد. همچنین، برای این زبان، ابزاری به نام اِکو ارائه شده که می‌تواند علاوه بر رفع برخی از ابهامات QVT-R، شرایطی را با استفاده از زبان قید شیء[38] روی فرامدل‌ها و تبدیل، تعریف و اعمال کند [20].

دستور زبان‌های سه‌تایی گراف[39] یک زبان قدرتمند بر پایه مفاهیم ریاضی برای تعریف تناظر میان دو نوع مدل به روش دوسویه و اعلانی می‌باشد [21]. با بهره‌مندی از شباهت میان گراف و فرامدل، سه گراف برای فرامدل مبدأ، فرامدل مقصد و فرامدل متناظر[40] در نظر گرفته می‌شود. فرامدل متناظر، زبانی را برای تعریف ارتباط میان مدل مبدأ و مقصد فراهم می‌کند. دستور زبان‌های سه‌تایی گراف برای تعریف ارتباط بین دو مدل، تبدیل یک مدل به مدل از نوع دیگر، محاسبه تناظر بین دو مدل یا نگهداری سازگاری دو مدل استفاده می‌شوند. این زبان، علاوه بر تبدیل مدل، در تجمیع و همگام‌سازی مدل‌ها نیز کاربرد دارد [5].

زبان تبدیل ژنوس[41] یک زبان مبتنی بر قید است که به طور ویژه برای پشتیبانی از مفهوم دوسویگی بنا شده است [22]. پیاده‌سازی آن بر پایه برنامه‌نویسی منطقی است و برای حل مسائل ان‌پی-سخت[42] مناسب است. یک تبدیل در این زبان، همانند تبدیل‌های QVT-R می‌تواند از نوع تحمیل یا بررسی باشد. اگر از نوع تحمیل باشد، در یک جهت خاص با انتخاب یکی از مدل‌ها به عنوان مقصد اجرا می‌شود. با اجرای فرآیند تبدیل، ابتدا برقراری سازگاری روابط بررسی می‌شود و برای آن‌هایی که سازگار نیستند، با ایجاد، حذف و تغییر مدل مقصد، سازگاری دو مدل برقرار می‌شود. این زبان از سازوکار پیشرفته ردیابی حمایت می‌کند که درآن همه عناصر چه آن‌هایی که درگیر تبدیل می‌شوند و چه آن‌هایی که از فرآیند تبدیل حذف می‌شوند، ذخیره می‌گردند. برتری این زبان در تبدیل دوسویه با مقایسه آن با گراف‌های سه‌تایی، برای برخی از کاربردها ثابت شده است [23].

زبان تبدیل شیءگرای دوسویه[43] یک روش رابطه‌ای بر اساس MOF است که فقط تبدیل‌های دوسویه یک‌به‌یکِ پوشا را پشتیبانی می‌کند [24]. تبدیل‌های یک‌به‌یکِ پوشا نوع خاصی از تبدیل‌های دوسویه هستند که تنها می‌توانند روابطی را پشتیبانی کنند که به ازای یک مدل مبدأ فقط و فقط یک مدل مقصد وجود داشته باشد و بالعکس.

زبان FunnyQT فضاهای نام زیادی همانند تبدیل درجا، تبدیل مدل تک‌سویه و دوسویه را فراهم می‌کند [25]. در تبدیل دوسویه، تناظری میان دو مدل که مدل چپ و مدل راست نامیده می‌شوند، با استفاده از روابط تبدیل (t-relations) تعریف می‌شود. هر رابطه می‌تواند شامل پیش‌شرط و پس‌شرط باشد. تبدیل‌های دوسویهٔ این زبان، برخلاف زبان ژنوس قطعی بوده و بسیار شبیه به QVT-R عمل می‌کنند. این زبان دارای نگاشت قابل ردیابی بسیار قوی همانند زبان ژنوس است و از این نظر که بر مبنای برنامه‌نویسی منطقی کار می‌کند، شباهت‌هایی با JTL دارد.

عدسی‌ها[44] برای اولین بار توسط فاستر[45] به عنوان یک چارچوب معنایی زبان برنامه‌نویسی[46] تعریف شد [26]. یک تبدیل دوسویه با استفاده از یک جفت عدسی، یعنی یک جفت تابع، تعریف می‌شود. تابع رو به جلو روی مدل مبدأ کار می‌کند و تابع معکوس، با استفاده از مدل مبدأ قبلی و مدل مقصد جدید، مدل مبدأ جدید را تولید می‌کند. در این حالت، توابع باید یک‌به‌یک باشند، بنابراین تبدیل بسیار محدود می‌شود. ایده فاستر روشی نامتقارن است که یکی از تابع‌ها به عنوان تابع اصلی و دیگری دید[47] آن در نظر گرفته می‌شود.

زبان جستجوی غیرساخت‌یافته[48] یک زبان پرس‌وجوی جبری گراف است که بر مبنای تبدیل‌های جبری گراف و زبان UnCAL شکل گرفته است [27]. این زبان برخلاف زبان QVT-R یک زبان تابعی بوده و با استفاده از آن می‌توان تبدیل‌ها را با هم ترکیب

---

[37] http://projects.ikv.de/qvt
[38] Object Constraint Language (OCL)
[39] Triple Graph Grammars (TGG)
[40] Corresponding metamodel
[41] Janus Transformation Language (JTL)
[42] NP-hard
[43] Bidirectional Object-Oriented Transformation Language (BOTL)
[44] Lenses
[45] Foster
[46] Programming language semantic framework
[47] View
[48] Unstructured Query Language (UnQL)

کرد. برای تضمین دوسویگی، از شباهت فرامدل و گراف استفاده کرده و فرامدل‌ها و مدل‌ها را بـه گراف‌هـای جبـری تبـدیل می‌کنـد. تبدیل‌های جبری گراف از مفهوم دوسویگی پشتیبانی کرده و در نتیجه، با تعریف تابع تبدیل رو به جلو در این زبان، اطلاعات ردیـابی و تابع تبدیل معکوس به صورت خودکار ساخته می‌شوند.

4. چارچوب اپسیلون

چارچوب اپسیلون زبان‌های متعددی را برای کار برروی مدل‌ها فراهم می‌کند که یکی از این زبان‌ها برای ارزیابی مدل در نظـر گرفتـه شده است. زبان اعتبارسنجی اپسیلون برای بیان قیود روی مدل‌هایی مبتنی بر فرامدل‌های مختلف به کار می‌رود کـه ارزیـابی درون-مدلی و بین- مدلی را سهولت می‌بخشد [28].

4-1 زبان اعتبارسنجی اپسیلون

هدف از طراحی زبان اعتبارسنجی اپسیلون رفع محدودیت‌های روش‌های اعتبارسنجی دیگر همانند زبان قید شیء می‌باشـد. از جملـه مزایای این زبان می‌توان به بازخورد بهتر به کاربر، شانس تعریف اخطار علاوه بر تعریـف خطـا، محـدود کـردن مجموعـه نمونـه‌هـا بـا استفاده از نگهبان ، اصلاح ناسازگاری‌ها با روش‌های متعدد، پشتیبانی از مدل‌سازی بینابینی اشاره کـرد. دو ویژگـی آخـر مـا را قـادر می‌سازد تا از این زبان برای تعریف تبدیل‌های دوسویه بهره ببریم. این زبان به‌صورت مجموعه‌ای از پیمانه‌ها[49] بیان می‌شود. هر پیمانه شامل مجموعه‌ای از قیود[50] می‌باشد که در بافت‌ها[51] دسته‌بندی می‌شوند. همچنین برای هر پیمانه می‌تـوان بلـوک‌هـای pre و post تعریف کرد که قبل و بعد از اجرای دستورات پیمانه، دستورات این بلوک‌ها اجرا می‌شود. در هر زمینه، یک نگهبان بـه طـور اختیـاری تعریف می‌شود که نمونه‌های مورد بررسی توسط قیودِ آن بافت را محدود می‌کند. در این حالت، اگر شرط آمده در نگهبان ارضا نشود، هیچ کدام از قیود تعریف شده در آن بافت مورد بررسی قرار نمی‌گیرند. نحو مربوط به بافت در زبان اعتبارسنجی اپسیلون در لیست 1 آمده است.

```
context <name> {
      (guard (:expression)|({statementBlock}))?
      (invariant)*
}
```

لیست (1): نحو بافت در زبان EVL [28]

هر قید می‌تواند در دو حالت خطا یا اخطار تعریف شود. ضرورت رفع اخطار از خطا کمتر است. در هر قید نیز می‌توان به طور اختیاری نگهبان تعریف کرد تا مجموعه نمونه‌های مورد بررسی را کوچک‌تر کرد. هر قید دارای دو قسمت بررسـی و اصـلاح اسـت. در قسـمت بررسی، یک شرط تعریف می‌شود. در صورت عدم ارضای شرط، پیامی به کاربر نمایش داده می‌شود کـه مـتن پیـام از قبـل در برنامـه آمده است و در آن دلیل رخداد خطا یا اخطار ذکر می‌شود. قسمت اصلاح شامل دستوراتی می‌شود که برای رفـع خطـا یـا اخطـار بـه وجود آمده نوشته شده‌اند. می‌توان چند راهکار برای رفع خطا ارائه کرد. نحو قید در لیست 2 قابل مشاهده است.

```
(constraint|critique) <name> {
      (guard (:expression)|({statementBlock}))?
      (check (:expression)|({statementBlock}))?
      (message (:expression)|({statementBlock}))?
      (fix)*
}
```

لیست (2): نحو قید در زبان EVL [28]

قسمت اصلاح خود شامل دو بخش است: بخش عنوان و بخش اجرایی. در بخش عنوان پیامی نوشته می‌شود تا بـه کـاربر بگویـد قـرار است چه کاری برای رفع خطا انجام گیرد. در بخش اجرایی دستوراتی که به رفع خطا منجر می‌شـوند، آورده مـی‌شـود. نحـو قسـمت اصلاح در لیست 3 آمده است.

---

[49] Module
[50] invariant
[51] Context

```
fix {
      (guard (:expression)|({statementBlock}))?
      (title (:expression)|({statementBlock}))?
      do{
            statementBlock
      }
}
```

لیست (3): نحو اصلاح در زبان EVL [28]

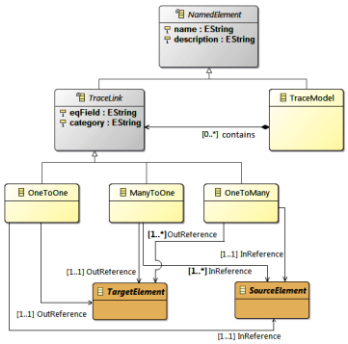
4-2 بحث و بررسی

ایده استفاده از زبان اعتبارسنجی اپسیلون برای تبدیل‌های دوسویه در ابتدا در [29] مطرح شد، اما این ایده در عمل چالش‌های متعددی دارد. یکی از این چالش‌ها این است که سازگاری تبدیل رو به جلو و رو به عقب باید با روش‌های وارسی گران‌قیمت بررسی شود. برای تعیین یک تبدیل دوسویه با استفاده از EVL باید برای هر عنصر در مدل مبدأ (مقصد) یک یا چند عنصر متناظر در مدل مقصد (مبدأ) وجود داشته باشد؛ در صورتی که این تناظر وجود نداشته باشد یک یا چند راه‌حل برای بازیابی ناسازگاری پیش آمده در بخش اصلاح مربوط به قید مورد بررسی پیشنهاد می‌شود. اما با استفاده از زبان EVL نمی‌توان تغییرات را به درستی انتشار داد. با توجه به مثال تبدیل بیان شده در بخش 2-3، اگر در مدل مقصد، نام ستون ID به StdNumber تغییر یابد، برنامه EVL آن را به عنوان ستون جدید در نظر می‌گیرد. در حالی که این صفت متناظر با این ستون در مدل مبدأ از قبل وجود داشته و کافی‌است فقط نام آن تغییر کند. در بخش 5، برای رفع این مشکل، روش پیشنهادی این مقاله را ارائه می‌دهیم.

5. تبدیل دوسویه با استفاده از EVL+trace

استفاده از زبان اعتبارسنجی اپسیلون به تنهایی [29] نمی‌تواند تغییرات را انتشار دهد، بلکه فقط می‌تواند عناصر را حذف یا اضافه کند. به عنوان یک راه‌حل، ما در [9] پیشنهاد کردیم که ارجاعاتی به عناصر متناظر مبدأ و مقصد در مدل سومی به نام مدل ردیابی ذخیره شود تا انتشار تغییر امکان‌پذیر باشد. بر اساس اصول تولید مدل‌گرا هر مدل مبتنی بر یک فرامدل است بنابراین مدل ردیابی نیز بر طبق فرامدل ردیابی نشان داده شده در شکل 5 ایجاد می‌شود. فرامدل ردیابی دارای ریشه‌ای به نام TraceModel است. این ریشه شامل تعدادی پیوند به نام TraceLink می‌باشد. بر اساس کاردینالیتی عناصر مبدأ و مقصد، هر پیوند می‌تواند از نوع یک به یک، یک به چند و چند به یک باشد. پیوند یک به یک برای ارجاع به یک عنصر مبدأ به یک عنصر مقصد، پیوند چند به یک برای ارجاع به چند عنصر مبدأ به یک عنصر مقصد و پیوند یک به چند برای اتصال یک عنصر مبدأ به چند عنصر مقصد به کار می‌رود.

هر پیوند برای کوچکترین بخش‌های مشترک (هم‌ارز) بین عناصر متناظر از مدل مبدأ و مقصد تعریف می‌شود. مقدار هم‌ارزی به صورت یک رشته در میدان هم‌ارزی (eqField) ذخیره می‌شود. همچنین صفت «رسته»[52] در TraceLink نمایانگر نوع قانون تبدیل است. برای مثال برای تبدیل نام رده به جدول از مقدار ClassToTableName و تبدیل نوع صفت به ستون را با مقدار AttributeToColumnDataType برای رسته انتخاب می‌شود. با استفاده از فرامدل پیشنهادی می‌توان عملیات حذف، اضافه و انتشار تغییر را امکان‌پذیر کرد. روش پیشنهادی بر مبنای زبان اعتبارسنجی اپسیلون و تکنیک‌های ردیابی به EVL+trace نام‌گذاری شده است. برای آن‌که نشان دهیم این روش چگونه کار می‌کند از محک Person2Person [10] استفاده کرده‌ایم.

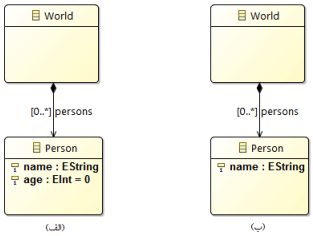


شکل (5): فرامدل ردیابی روش پیشنهادی مربوط به EVL+trace [9]

[52] category

5-1 محک Person2Person

این مثال برای محک و نمایش پیوندهای یک به یک در مدل ردیابی در این مقاله استفاده می‌شود که در آن فرامدل‌های مبدأ (left) و مقصد (right) بسیار ساده طراحی شده‌اند. در شکل 6، تصویری از هر دو فرامدل نمایش داده شده است.



شکل (6): فرامدل‌های Person2Person : (الف)- فرامدل left، (ب)- فرامدل right

فرامدل left دارای یک دنیا شامل تعدادی فرد است که با یک فرارده world و فرارده Person نشان داده می‌شود. هر فرد دارای دو صفت است: نام (name) از نوع رشته‌ای و سن (age) از نوع عدد صحیح؛ صفت name یک شناسه برای فرد محسوب می‌شود به این معنی که هر شیء از نوع Person باید دارای نام منحصربه‌فرد باشد. زبان فرامدل left به صورت مجموعه‌ای از توابع جزئی تعریف می‌شود به طوری‌که مدل‌های منطبق بر این فرامدل بر اساس فرمول (1) تعریف می‌شود:

$$model^{left}: Int \rightarrow String \times Int \qquad (1)$$

فرامدل right شامل مجموعه‌ای از افراد (Person) می‌شود که هر فرد دارای فقط یک صفت نام (name) می‌باشد. در این فرامدل نیز صفت name یک صفت کلیدی و منحصربه فرد برای هر شیء از نوع Person است. زبان فرامدل مقصد مجموعه‌ای از توابع جزئی می‌باشد که به صورت فرمول (2) تعریف می‌شوند:

$$model^{right}: Int \rightarrow String \quad (2)$$

در صورتی‌که نام یک فرد در یکی از مدل‌ها تغییر یابد، باید بتوان فرد متناظر با آن را در مدل دیگر پیدا کرد و نام او را تغییر داد (انتشار تغییر). در این مثال، سه عملیات ممکن باید توسط هر روش تبدیل دوسویه امکان‌پذیر باشد:

1- حذف: اگر کاربر یکی از افراد را از مدل left (right) حذف کند، فرد متناظر در مدل right (left) باید حذف شود تا سازگاری برقرار شود.

2- تغییر: اگر کاربر نام یکی از افراد را در مدل left (right) تغییر دهد، باید نام فرد متناظر با آن در مدل right (left) توسط تبدیل دوسویه به‌روزرسانی شود. توجه کنید که تغییر در سن یک فرد از مدل left هیچ تاثیری روی مدل right ندارد.

3- اضافه کردن: اگر کاربر یک فرد جدید را به مدل left (right) اضافه کند باید در مدل دیگر، یک فرد با همان نام ایجاد شود. در صورتی‌که فرد جدید در مدل right اضافه شده باشد و تبدیل بخواهد برای برقراری سازگاری فردی متناظر با آن را در مدل left ایجاد کند، سن پیش‌فرض 18 در نظر گرفته می‌شود.

در محک Person2Person نمونه‌های آزمونی در نظر گرفته شده است که روش پیشنهادی تبدیل دوسویه را می‌توان بر روی آن‌ها به محک‌زنی گذاشت. این نمونه‌ها برای آزمون سه نوع ویژگی دوسویه که عبارتند از صحت[53]، جامعیت[54] و بقراط‌گونگی[55] به کار می‌روند. **صحت**، یعنی عمل تبدیل بتواند سازگاری را بین دو مدل برقرار کند. **جامعیت**، یعنی عمل تبدیل بتواند برای همه ورودی‌های معتبر قابل انجام باشد. **بقراط‌گونگی**، یعنی هرگاه دو مدل مبدأ و مقصد بر اساس تعریف تبدیل سازگار بودند بعد از اجرای تبدیل هیچ تغییری در دو مدل ایجاد نشود. به عبارت دیگر، این ویژگی از تغییرات غیرضروری در مدل‌ها جلوگیری می‌کند. این نمونه‌ها در جدول 1 همراه با توضیحات نشان داده شده‌اند.

برای نمونه‌های BCF و BCB، 4 زیرنمونه تعریف می‌شود. مدل خالی، مدل دو نفره، مدل 8 نفره و مدل 3 نفره که نام افراد در آن صرفاً از نظر حروف بزرگ و کوچک انگلیسی متفاوت است. مدل آخر، برای بررسی حساس بودن روش دوسویه روی حروف در نظر گرفته شده است. در MCB، دو حالت در نظر گرفته می‌شود: یکی حالتی که مدل مبدأ و مقصد خالی باشند و باید روش دوسویه

---

[53] Correctness
[54] Completeness
[55] Hippocracticness

بدون هیچ تغییری در مدل‌ها، سازگاری را تشخیص دهد، در حالت دیگر، مدل مبدأ خالی و مدل مقصد دو نفره به روش دوسویه داده می‌شود و درستی تبدیل رو به عقب تضمین می‌شود.

جدول (1): نمونه‌های آزمون Person2Person

نام مورد آزمون	توضیحات
Person2Person.BCF	هدف از این نمونه ارزیابی ویژگی‌های دوسویه صحت و جامعیت برای تبدیل رو به جلو در حالت دسته‌ای یعنی حالتی که از قبل هیچ مدل مقصدی (right) وجود ندارد و قرار است با اجرای تبدیل از روی مدل مبدأ ایجاد شود. در این حالت، مدل right دارای شیء از نوع Person نیست.
Person2Person.BCB	این نمونه همانند نمونه قبلی بوده با این تفاوت که برای تبدیل رو به عقب در نظر گرفته شده است، به طوری‌که مدل left خالی است و باید از روی مدل right ایجاد شود.
Person2Person.MCB	هدف از این نمونه بررسی ویژگی‌های دوسویه صحت، جامعیت و بقراط‌گونگی می‌باشد در حالتی که هر دو مدل left و right وجود دارند اما مدل right تغییر کرده است و باید بتوان این تغییر را در مدل left انتشار داد. به طوری‌که اگر نام یک فرد در مدل right عوض شود، نام فرد متناظر در مدل مبدأ تغییر یابد اما سن او بدون تغییر باقی بماند. تغییر سن یک تغییر غیرضروری محسوب می‌شود.

5-2 تبدیل دوسویه EVL+trace روی محک Person2Person

برای پشتیبانی از عمل انتشار تغییر، مدل ردیابی در EVL+trace باید شناسه افراد متناظر را در یک پیوند یک به یک ذخیره کند، به طوری که مقدار صفت رستهٔ آن پیوند را PersonToPersonName قرار می‌دهیم. برای تعریف یک شناسه برای هر شیء Person در هر دو مدل باید ویژگی xmi:id را برای هر عنصر در هر دو مدل فعال کنیم. این فعال‌سازی با یک دستور ساده در اپسیلون بدون هیچ اثر جانبی در فرامدل‌ها امکان‌پذیر است: *model's name.resource.useXmiIds =true*

این دستور را باید در بلوک pre برنامه نوشت تا قبل از همه دستورات اجرا شود. از آن‌جایی‌که برنامه روی سه مدل مبدأ، مقصد و ردیابی اجرا می‌شود، کد این قسمت به صورت لیست 4 نشان داده می‌شود.

```
1 pre{
2     EVLTrace.resource.useXmiIds= true;
3     Left.resource.useXmiIds= true;
4     Right.resource.useXmiIds= true;}
```

لیست (4): بلوک pre برنامه EVL+trace

مقدار eqField در پیوند ردیابی مذکور همان نام مشترک دو فرد متناظر از دو مدل خواهد بود. هرگاه یک شیء Person در مدل right از روی شیئی از مدل left ایجاد شود، یک پیوند در مدل ردیابی به صورت خودکار ایجاد می‌شود که ارجاعاتی را به شناسه‌های هر دو شیء و نام مشترک میان آن‌ها را در خود ذخیره می‌کند. حال برای انجام سه عمل مذکور با استفاده از روش پیشنهادی بر اساس جدول 2 عمل می‌کنیم.

در جدول 2، نماد l نمایان‌گر نام فرد در مدل left، r نشان‌دهنده‌ی نام فرد متناظر در مدل right و t بیان‌گر مقدار میدان eqField در پیوند ردیابی متناظر از مدل ردیابی می‌باشد. به شکل 7 توجه کنید.

شکل (7): مثالی از عناصر متناظر مبدأ، مقصد و ردیابی

بر اساس جدول 2، حالت 1 حالتی را نشان می‌دهد که کاربر هیچ تغییری در مدل مبدأ و مقصد ایجاد نکرده و در نتیجه، نام افراد متناظر از مدل مبدأ و مقصد با مقدار میدان هم‌ارزی پیوند ردیابی یکسان می‌باشد. در این حالت، روش پیشنهادی به دلیل برقراری سازگاری مدل‌ها از قبل هیچ عملیاتی را انجام نمی‌دهد.

جدول (2): عملکرد روش پیشنهادی روی محک Person2Person

حالت	Left	trace	right	شرط بررسی	عمل کاربر	عمل EVL+trace
1	l	t	r	$l = t = r$	–	–
2	l	t	-	$l = t$	حذف مقصد	حذف مبدأ و پیوند
3	l	t	-	$l \neq t$	حذف مقصد و تغییر مبدأ	حذف مبدأ و پیوند
4	-	t	r	$r = t$	حذف مبدأ	حذف مقصد و پیوند
5	-	t	r	$r \neq t$	حذف مبدأ و تغییر مقصد	حذف مقصد و پیوند
6	-	t	-	-	حذف مبدأ و مقصد	حذف پیوند
7	l	t	r	$l \neq t \wedge t = r$	تغییر مبدأ	تغییر مقصد و پیوند
8	l	t	r	$l = t \wedge t \neq r$	تغییر مقصد	تغییر مبدأ و پیوند
9	l	t	r	$l \neq t \wedge t \neq r \wedge l = r$	تغییر مبدأ و مقصد به صورت سازگار	تغییر پیوند
10	l	t	r	$l \neq t \neq r$	تغییر مبدأ و مقصد به صورت ناسازگار	تغییر عناصر بر مبنای انتخاب کاربر
11	l	-	-	-	اضافه کردن مبدأ	اضافه کردن مقصد و پیوند
12	-	-	r	-	اضافه کردن مقصد	اضافه کردن مبدأ و پیوند
13	l	-	r	$l = r$	اضافه کردن مبدأ و مقصد به صورت سازگار	اضافه کردن پیوند
14	l	-	r	$l \neq r$	بی معنا	–

5-2-1 حذف

قطعه کد مربوط به بررسی و اعمال عمل حذف در لیست 5 آمده است. حالات 2 تا 6 از جدول 2 نشان می‌دهند که کاربر یکی از عناصر مبدأ یا مقصد را حذف کرده و در نتیجه، انتظار داریم برنامه EVL+trace نیز اشیاء متناظر با عنصر حذف شده را پاک کند. توجه شود که عمل حذف نسبت به عمل تغییر ارجح بوده، بنابراین در حالات 3 و 5 با وجود رخداد عمل تغییر، برای برقراری سازگاری عمل تغییر انتخاب می‌شود.

برای بررسی این‌که آیا عمل حذفی رخ داده یا خیر، کافی است بررسی شود که آیا پیوندی وجود دارد که ارجاع چپ یا ارجاع راست آن خالی باشد. حالات 2 و3 نشان می‌دهند که ارجاع راست خالی است؛ حالات 4 و 5 بیان می‌کنند که ارجاع چپ خالی شده است. حالت 6 نشان می‌دهد که هر دو ارجاع خالی هستند؛ به عبارت دیگر، کاربر هر دو فرد متناظر را از مدل مبدأ و مقصد حذف کرده است. برای بازیابی سازگاری انتظار داریم که روش پیشنهادی، عناصر باقی‌مانده در مدل‌های دیگر را نیز حذف کند.

```
context EVLTrace!OneToOne{
    guard: self.isPersonPerson()
    constraint RightPersonRemoval{
        check: not self.isLeftRemoved()
        message: "Removal: Person "+self.eqField+
                " has been removed from Left"
        fix{
            title: "Remove Person "+self.eqField+" from Right"
            do{
                var rperson = Right!Person.all.
                        select(p|p.id() = self.target.id()).first;
                if (rperson.isDefined()) delete rperson;
                delete self;}
        }}
    constraint LeftPersonRemoval{
        guard : not self.isLeftRemoved()
        check: not self.isRightRemoved()
        message: "Removal: Person "+self.eqField+
                " has been removed from Right"
        fix{
            title: "Remove Person "+self.eqField+" from Left"
            do{
                var lperson = Left!Person.all.
                        select(p|p.id() = self.source.id()).first;
                if (lperson.isDefined()) delete lperson;
                delete self;}
        }}
}
```

لیست (5): قطعه کد مربوط به بررسی و اعمال عمل حذف

توجه شود که در قطعه کد مربوط به حذف، توابعی استفاده شده است که بدنه‌ی آن‌ها در لیست 6 آمده است.

```
operation EVLTrace!OneToOne isRemovable():Boolean{
    if(self.isLeftRemoved() or self.isRightRemoved()) return true;
    return false;}

operation EVLTrace!OneToOne isLeftRemoved():Boolean{
    if(not self.source.isTypeOf(Left!Person)) return true;
    return false;}

operation EVLTrace!OneToOne isRightRemoved():Boolean{
    if(not self.target.isTypeOf(Right!Person)) return true;
    return false;}
```

لیست (6): قطعه کد مربوط به بدنه توابع مورد استفاده در عملیات حذف

5-2-2 تغییر

پس از بررسی حذف نوبت به بررسی تغییر می‌رسد. در لیست 7 قطعه کد مربوط به اننتشار تغییر نشان داده شده است. اگر کاربر نـام یک یا چند فرد را در مدل‌ها تغییر دهد، این مسئله با بررسی پیوندهای ردیابی امکان‌پذیر است. در ابتدا باید نگهبانی تعریف کـرد کـه فقط نمونه‌هایی بررسی شوند که عنصر متناظر آن‌ها حذف نشده است (زیرا عمل حذف بر عمل تغییر ارجح است). تغییر در نـام یـک فرد از مقایسه مقدار eqField در پیوند با نام آن فرد قابل کشف خواهد بود زیرا پیوندهای ردیـابی همیشـه مقـادیر قبـل از تغییـر در مدل‌های مبدأ و مقصد را نگه می‌دارند. پس از کشف تغییر، نام فرد متناظر در مدل دیگر و مقدار eqField پیوند ردیابی برای بازیـابی سازگاری بین دو مدل تغییر می‌یابد.

در جدول 2، حالات 7 تا 10 نشان‌دهنده عمل تغییر هستند. در حالت 7، نام فرد در مدل مبدأ تغییر کرده است زیرا با مقدار مشترک در eqField متفاوت است ($l \neq t$)، اما نام فرد در مدل مقصد تغییری پیدا نکرده است ($t = r$). بنابراین، انتظار می‌رود که تغییر نام در مدل مبدأ به مدل مقصد و مدل ردیابی نیز انتشار یابد. برعکس، حالت 8 نشان می‌دهد که نام فرد در مدل مقصد تغییر یافتـه ($t \neq r$) اما در مدل مبدأ ثابت مانده است ($l = t$). در نتیجه، تغییر از r به l و t انتشار می‌یابد. حالت 9، حالتی را نمایش مـی‌دهـد کـه هـر دو فرد در مدل مبدأ و مقصد، نامشان توسط کاربر به نامی یکسان تغییر یافته اما از آن‌جایی که eqField نـام قبلـی را نشـان مـی‌دهـد، مقدار t با مقادیر l و r مساوی نیست. در نتیجه، کافی است نام مشترک مربوط به l و r را بـه t انتشـار داد. در نهایـت، در حالـت 10 کاربر نام هر دو فرد مبدأ و مقصد را تغییر داده به طوری که این نام‌ها یکسان نیستند. حالت 10 نشان‌دهنده تداخلی اسـت کـه بـرای رفع آن باید تصمیم کاربر را مدنظر قرار داد. به عبارت دیگر، کاربر باید تعیین کند که نام مبدأ را بر مقصد برتری بدهد یا بالعکس.

```
context EVLTrace!OneToOne
{
    guard: self.isPersonPerson() and not self.isRemovable()
    constraint RightPersonModification{
        check: self.source.name = self.eqField
        message: "Modification: Person "+self.eqField+
            " in the Left has been modified to "+self.source.name
        fix{
            title: "Modify the name of Person "+self.eqField+
                " in the Right"
            do{
                var rperson = Right!Person.all.
                    select(p|p.id() = self.target.id()).first;
                rperson.name = self.source.name;
                self.name = self.source.name;
                self.eqField = self.source.name;}
        }}
    constraint LeftPersonModification{
        check: self.target.name = self.eqField
        message: "Modification: Person "+self.eqField+
        " in the Right has been modified to"+self.target.name
        fix{
            title: "Modify the name of Person "+self.eqField+
                " in the Left"
            do{
                var lperson = Left!Person.all.
                    select(p|p.id() = self.source.id()).first;
                lperson.name = self.target.name;
                self.name = self.target.name;
                self.eqField = self.target.name;}
        }}
}
```

لیست (7): قطعه کد مربوط به بررسی و انتشار تغییر

5-2-3 اضافه کردن

اگر کاربر یک عنصر جدید را در یک یا هردو مدل اضافه کند باید شیء جدیدی متناظر با آن در مدل دیگر اضـافه شـود. حـالات 11، 12 و 13 برای بررسی این عمل شکل گرفته‌اند. در لیست 8، قطعه کد مربوط به حالت 11 و 13 دیده می‌شود.

برای تشخیص اشیاء جدید کافی است بررسی کنیم که شناسه شیء توسط هیچ یک از پیونـدهای ردیـابی مـورد ارجـاع قـرار نگرفتـه است. بنابراین، همان‌طور که در جدول 2 دیده می‌شود، در این سه حالت نماد t وجود ندارد یعنی عناصری مورد بررسی قرار می‌گیرند که توسط هیچ پیوندی ردیابی نشده‌اند. حالت 11 بیان می‌کند که برای هر فرد جدید از مدل left هیچ فرد هم‌نام متناظری در مـدل دیگر وجود نداشته باشد. در این حالت، علاوه بر ایجاد عنصر متناظر با شیء جدید در مدل right باید یک پیوند ردیـابی میـان آن دو ایجاد کرد.

```
context Left!Person
{
    guard: self.isNew()
    constraint RightPersonExist{
        check: not self.hasEquivalentPerson()
        message: "Addition: Left Person "+self.name+
            " has an equivalent in the Right but aren't linked"
        fix{
            title:"Link two Persons with name "+self.name
            do{
                var rperson= self.getEquivalentPerson();
                self.createPersonPersonTrace(rperson);}
            }}
    constraint RightPersonAddition{
        check: self.hasEquivalentPerson()
        message: "Addition: Left Person "+self.name+
            " has no equivalent in the Right"
        fix{
            title:"Create an equivalent Person with name "+
                self.name+" in the Right"
            do{
                var rperson : new Right!Person;
                rperson.name = self.name;
                var rworld = Right!World.all.first;
                rworld.persons.add(rperson);
                self.createPersonPersonTrace(rperson);}
            }}
}
```

لیست (8): قطعه کد مربوط به حالات 11 و 13 برای عمل اضافه کردن

حالت 12 برعکس حالت 11 نشان می‌دهد که عنصر جدیدی در مدل right تشخیص داده شده که هیچ فردی در مدل مبدأ متناظر با آن نیست. در حالت 13، برای هر فرد جدید در مدل left، کاربر خود یک فرد جدید هم‌نام با آن در مدل right اضافه کرده بوده است اما هیچ پیوند بین آن‌ها وجود ندارد؛ در این حالت، کافی است فقط یک پیوند ردیابی که به عناصر جدید از هر دو مدل اشاره می‌کند، ایجاد کرد. همان‌طور که دیده می‌شود قطعه کد مربوط به حالات 11 و 13 از توابع hasEquivalentPerson، getEquivalentPerson و createPersonPersonTrace استفاده می‌کند که در لیست 9 و 10، بدنه آن‌ها نمایش داده شده است.

```
operation Left!Person hasEquivalentPerson():Boolean{
    var rperson = self.getEquivalentPerson();
    if(rperson.isDefined()) return true;
    else return false;}

operation Left!Person getEquivalentPerson():Right!Person{
    return Right!Person.all.select(p|p.name=self.name).first;}
```

لیست (9): بدنه توابع hasEquivalentPerson و getEquivalentPerson

تابع اول از لیست 9، در صورتی‌که فرد جدید از مدل مبدأ دارای عنصر متناظر در مدل مقصد باشد، مقدار *true* را بازمی‌گرداند. تابع دوم، شیء متناظر با فرد جدید از مدل left را بازمی‌گرداند. در لیست 10، تابع createPersonPersonTrace، یک پیوند را بین شیء فراخوانی‌کننده و شیء ارسالی به عنوان پارامتر در مدل ردیابی ایجاد می‌کند.

```

operation Left!Person createPersonPersonTrace(person:Right!Person)
{   var trace : new EVLTrace!OneToOne;
    var tracemodel = EVLTrace!MModelRoot.all.first;
    tracemodel.relations.add(trace);
    trace.category = "PersonToPerson";
    trace.name = self.name;
    trace.eqField = self.name;
    trace.source = self;
    trace.target = person;}
```

لیست (10): قطعه کد مربوط به تابع createPersonPersonTrace

لیست 11 قطعه کد مربوط به حالت 12 را نشان می‌دهد.

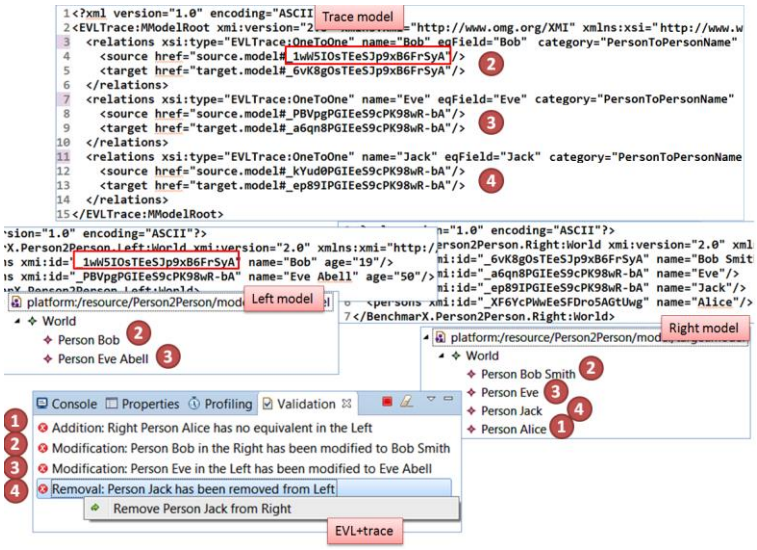
```
context Right!Person{
    guard: self.isNew()
    constraint LeftPersonAddition{
        check: self.hasEquivalentPerson()
        message: "Addition: Right Person "+self.name+
            " has no equivalent in the Left"
        fix{
            title:"Create an equivalent Person with name "+
                self.name+" in the Left"
            do{
                var lperson : new Left!Person;
                lperson.name = self.name;
                lperson.age = 18;
                var lworld = Left!World.all.first;
                lworld.persons.add(lperson);
                lperson.createPersonPersonTrace(self);}
        }}
}
```

لیست (11): قطعه کد مربوط به حالت 12 برای عمل اضافه کردن

5-3 نتایج اجرای EVL+trace روی نمونه‌های آزمون Person2Person

شکل 8 نمونه‌ای از اجرای روش پیشنهادی روی تبدیل Person2Person را نشان می‌دهد. هر پیوند ردیابی دارای دو ارجاع است که شامل xmi:idهایی به عناصر مبدأ و مقصد می‌شود. طبق شکل 8، سه پیوند ردیابی برای افراد Bob، Eve و Jack در مدل ردیابی وجود دارد. به علاوه، یک فرد جدید به نام Alice در مدل right اضافه شده که در مدل ردیابی هیچ پیوندی برای آن نیست.

همچنین نام Bob در مدل right به Bob Smith و نام Eve در مدل left به Eve Abell تغییر یافته و فرد با نام Jack از مدل left حذف شده است.



شکل (8): تصویر لحظه‌ای از اجرای EVL+trace روی یک مثال از Person2Person

در جدول 3، نتایج پیاده‌سازی EVL+trace روی نمونه‌های مورد آزمون محک Person2Person نمایش داده شده است.

جدول (3): نتایج روش پیشنهادی روی نمونه‌های آزمون Person2Person

✓ = قابل‌پشتیبانی، ✗ = غیرقابل‌پشتیبانی، - = تعریف نشده برای نمونه

نمونه‌های آزمون	صحت	جامعیت	بقراط‌گونگی
Person2Person.BCF.1	✓	✓	-
Person2Person.BCF.2	✓	✓	-
Person2Person.BCF.3	✓	✓	-
Person2Person.BCF.4	✓	✓	-
Person2Person.BCB.1	✓	✓	-
Person2Person.BCB.2	✓	✓	-
Person2Person.BCB.3	✓	✓	-
Person2Person.BCB.4	✓	✓	-
Person2Person.MCB.1	✓	✓	✓
Person2Person.MCB.2	✓	✓	✓

همان‌طور که می‌بینیم، روش پیشنهادی روی موارد آزمون به خوبی عمل می‌کند. باید توجه کرد که روش EVL+trace دارای یک اجرای تعاملی است که در آن از کاربر پرسیده می‌شود که یکی از راهکارهای پیشنهادی را برای اصلاح ناسازگاری برگزینند. اما می‌توان کد تبدیل را طوری نوشت که به غیر از موارد تداخل که نیازمند تصمیم کاربر هستند، در بقیه موارد اصلاح ناسازگاری به صورت خودکار انجام گیرد. همچنین روش EVL+trace علاوه بر آن که به عنوان یک روش تبدیل دوسویه در نظر گرفته می‌شود، می‌تواند به عنوان یک همگام‌ساز نیز عمل کند. زیرا در آن تبدیل‌های رو به جلو و رو به عقب همزمان در یک بار اجرای برنامه EVL انجام می‌پذیرند. در حالی‌که اگر فقط از EVL بدون مدل ردیابی برای تبدیل دوسویه استفاده شود، همگام‌ساز در برخی موارد نمی‌تواند به خوبی عمل کند. برای مثال، در حالتی که نام فردی در مدل مقصد تغییر می‌کند، EVL بدون مدل ردیابی آن را یک شیء جدید در نظر می‌گیرد و در مدل مبدأ یک فرد با هم‌نام با نام جدید ایجاد می‌کند، به‌طوری‌که، سن آن را مساوی با 18 تنظیم می‌کند. اما فرد متناظر قبلی در مدل مبدأ حذف می‌شود که ممکن است سن او متفاوت با عدد 18 باشد. در این حالت، بقراط‌گونگی زیر سوال می‌رود و همگام‌سازی بین مدل‌ها به درستی امکان‌پذیر نخواهد بود.

5-4 بررسی ویژگی‌ها و قابلیت‌های روش EVL+trace

روش EVL+trace پیشنهاد می‌دهد که همه قیود مربوط به بررسی و اعمال سازگاری مربوط به تبدیل رو بـه جلـو و رو بـه عقـب در یک برنامه اعتبارسنجی اپسیلون تعریف شود. در این حالت، هر دو تبدیل در یک زمان می‌توانند اجرا شوند و در صورت تـداخل میـان آن‌ها، پیامی به کاربر نمایش داده می‌شود و راهکارهایی برای حل تداخل به او ارائه می‌گردد. کـاربر از بـین راهکارهـا یکـی را انتخـاب می‌کند و پس از آن، به صورت خودکار تداخل رفع می‌گردد. بنابراین، این روش می‌تواند علاوه بر اجرای تبدیل دوسویه به عنوان یـک همگام ساز شناخته شود. روش پیشنهادی، اولین روش تعاملی در مقایسه با روش‌های تبدیل مدل دوسویه [7] اسـت کـه بـه صـورت عملی ویژگی‌های انتشار تغییر و بقراط‌گونگی را پشتیبانی می‌کند، در حالی‌که اکثر روش‌های دوسویه این دو ویژگی را ندارند [5, 6].
همان‌طور که در پیاده‌سازی‌های این مقاله نشان داده شد، این روش هر سـه عملیـات بـه‌روزرسـانی حـذف، تغییـر و اضـافه کـردن را پشتیبانی می‌کند، در حالی‌که اکثر همگام‌سازها [7]، عمل اضافه کردن - مهم‌ترین عمل برای تبدیل- را نمـی‌توانـد حمایـت کنـد. در مقایسه با روش زبان تبدیل شیءگرای دوسویه [24]، EVL+trace می‌تواند سناریوهای تبدیل غیریک به یک را نیز پشتیبانی کند. از آن‌جایی که نحو زبان اعتبارسنجی اپسیلون بسیار شبیه به زبان قید شیء -زبان شـناخته شـده بـرای همـه تولیدکننـدگان مـدل‌گرا- می‌باشد، کار با روش ما راحت‌تر از روش‌هایی همانند دستور زبان‌هـای سـه‌تـایی گـراف [5, 21] - کـه دارای یـک نمادسـازی خـاص هستند- می‌باشد. به دلیل آن‌که چارچوب اپسـیلون بـا همـه نسـخه‌هـای اکلیـپس تـا جدیـدترین آن سـازگار اسـت، قابلیـت حمـل EVL+trace بسیار بالاست؛ در حالی‌که ابزارهای QVT-R و TGG [6, 21] یا منسوخ شده‌اند و یا پشتیبانی کـاملی از ویژگـی‌هـای دوسویگی ندارند.

6. جمع‌بندی

در این مقاله، روش EVL+trace بر مبنای زبان اعتبارسنجی چارچوب اپسیلون و تکنیک‌های ردیابی‌پذیری به صورت مشروح ارائه و پیاده‌سازی شد. برای نمایش قابلیت‌های دوسویگی و چگونگی استفاده از این روش آن را روی محک Person2Person پیاده‌سـازی کردیم. نتایج ارزیابی روش روی این محک نشان می‌دهد که قابلیت‌های دوسویگی صحت، جامعیت و بقراط‌گونگی به طـور کامـل روی نمونه‌های آزمون حمایت می‌شوند. بنابراین، روش پیشنهادی می‌تواند به ازای هر مدل ورودی یک مدل خروجی ارائه دهد (جامعیـت) به طوری‌که آن مدل خروجی سازگار با مدل ورودی باشد (صحت). به دلیل بررسی قبـل از اعمـال تبـدیل، EVL+trace از تغییـرات غیرضروری روی مدل‌ها جلوگیری می‌کند و در نتیجه، بقراط‌گونگی را نیز محقق می‌سازد.
همچنین روش این مقاله برای تحقق تبدیل دوسویه و همگام‌سازی بین مدل‌ها سه عملیات حذف، تغییر و اضافه کردن مربوط بـه هـر دو سوی تبدیل را در یک برنامه EVL تعریف می‌کند تـا در حـین پشـتیبانی ویژگـی انتشـار تغییـر بتوانـد در یـک زمـان تـداخل و ناسازگاری‌های بین دو مدل را رفع سازد. این روش، تصمیمات کاربر را برای رفع تداخل در نظر می‌گیرد و از این رو بـه عنـوان اولـین روش تعاملی دوسویه شناخته شده است. برای بالا بردن سطح تجرید روش EVL+trace، به عنوان کارهای آینده پیشـنهاد مـی‌شـود که از تکنیک‌های تولید مدل‌گرا استفاده شود تا کد برنامه EVL به صورت خودکار تولید و پیاده‌سازی شود. در این حالت، تولیدکننده می‌تواند با ایجاد مدل تبدیل در سطح انتزاع بالاتر، برنامه تبدیل دوسویه را با استفاده از یک تولیدکننده کـد ایجـاد کنـد و آن را روی مدل‌های متعدد اجرا نماید.

مراجع

[1] C. Atkinson and T. Kühne, "*Model-driven development: a metamodeling foundation," Software, IEEE*, vol. 20, no. 5, pp. 36–41, 2003.

[2] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2012.

[3] B. Zamani, *On verifying the use of a pattern language in model driven design*. PhD thesis, Concordia University, July 2009.

[4] S. Sendall and W. Kozaczynski, *"Model Transformation: The Heart and Soul of Model-Driven Software Development," Software, IEEE*, vol. 20, no. 5, pp. 42–45, 2003.

[5] H. Giese and R. Wagner, *"From model transformation to incremental bidirectional model synchronization,"Software & Systems Modeling*, vol. 8, no. 1, pp. 21–43, 2009.

[6] P. Stevens, *"Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions,"* in *Model Driven Engineering Languages and Systems* (G. Engels, B. Opdyke, D. Schmidt, and F. Weil, eds.), vol. 4735 of *Lecture Notes in Computer Science*, pp. 1–15, Springer Berlin Heidelberg, 2007.

[7] S. Hidaka, M. Tisi, J. Cabot, and Z. Hu, *"Feature-based classification of bidirectional transformation approaches,"* Software & Systems Modeling, pp. 1–22, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10270-014-0450-0

[8] A. Anjorin, S. Rose, F. Deckwerth, and A. Schrr, *"Efficient model synchronization with view triple graph grammars,"* in Modelling Foundations and Applications, ser. Lecture Notes in Computer Science, J. Cabot and J. Rubin, Eds. Springer International Publishing, 2014, vol. 8569, pp. 1–17. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-09195-2 1

[9] L. Samimi-Dehkordi, B. Zamani, and S. Kolahdouz-Rahimi, *"From trace-based inter-model validation to bidirectional model synchronization with reconciliation,"* in *the 5th International eConference on Computer and Knowledge Engineering (ICCKE 2015),* Mashhad, Iran, October, 2015, pp. 123-130, 2015.

[10] A. Anjorin, A. Cunha, H. Giese, F. Hermann, A. Rensink, and A. Schürr, *"BenchmarX,"* in *Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference (EDBT/ICDT 2014)*, Athens, Greece, March 28, 2014, pp. 82–86, 2014.

[11] B. Hailpern and P. Tarr, *"Model-driven development: The good, the bad, and the ugly," IBM Systems Journal*, vol. 45, no. 3, pp. 451–461, 2006.

[12] D. Ameller, *"Considering Non-Functional Requirements in Model-Driven Engineering,"* Master's thesis, Llenguatges i Sistemes Informàtics (LSI), June 2009.

[13] J. Bézivin, *"In Search of a Basic Principle for Model Driven Engineering," UPGRADE – The European Journal for the Informatics Professional*, vol. 5, no. 2, pp. 21–24, 2004.

[14] S. Balsamo, A. di Marco, P. Inverardi, and M. Simeoni, *"Model-based performance prediction in software development: a survey," Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 295–310, 2004.

[15] K. Czarnecki, J. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. Terwilliger, *"Bidirectional Transformations: A Cross-Discipline Perspective,"* in *Theory and Practice of Model Transformations* (R. Paige, ed.), vol. 5563 of *Lecture Notes in Computer Science*, pp. 260–283, Springer Berlin Heidelberg, 2009.

[16] P. Stevens, *"A Landscape of Bidirectional Model Transformations,"* in *Generative and Transformational Techniques in Software Engineering II* (R. Lämmel, J. Visser, and J. Saraiva, eds.), vol. 5235 of *Lecture Notes in Computer Science*, pp. 408–424, Springer Berlin Heidelberg, 2008.

[17] P. Stevens, *"Bidirectionally Tolerating Inconsistency: Partial Transformations,"* in *Fundamental Approaches to Software Engineering* (S. Gnesi and A. Rensink, eds.), vol. 8411 of *Lecture Notes in Computer Science*, pp. 32–46, Springer Berlin Heidelberg, 2014.

[18] I. Kurtev, *"State of the Art of QVT: A Model Transformation Language Standard,"* in *Applications of Graph Transformations with Industrial Relevance* (A. Schürr, M. Nagl, and A. Zündorf, eds.), vol. 5088 of *Lecture Notes in Computer Science*, pp. 377–393, Springer Berlin Heidelberg, 2008.

[19] E. Guerra and J. de Lara, *"An Algebraic Semantics for QVT-Relations Check-only Transformations," Fundamenta Informaticae*, vol. 114, no. 1, pp. 73–101, 2012.

[20] N. Macedo and A. Cunha, *"Implementing QVT-R Bidirectional Model Transformations Using Alloy,"* in *Fundamental Approaches to Software Engineering* (V. Cortellessa and D. Varró, eds.), vol. 7793 of *Lecture Notes in Computer Science*, pp. 297–311, Springer Berlin Heidelberg, 2013.

[21] A. Schürr and F. Klar, *"15 Years of Triple Graph Grammars,"* in *Graph Transformations* (H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, eds.), vol. 5214 of *Lecture Notes in Computer Science*, pp. 411–425, Springer Berlin Heidelberg, 2008.

[22] A. Cicchetti, D. Ruscio, R. Eramo, and A. Pierantonio, *"JTL: A Bidirectional and Change Propagating Transformation Language,"* in *Software Language Engineering* (B. Malloy, S. Staab, and M. Brand, eds.), vol. 6563 of *Lecture Notes in Computer Science*, pp. 183–202, Springer Berlin Heidelberg, 2011.

[23] R. Eramo and A. Bucaioni, *"Understanding bidirectional transformations with TGGs and JTL," Electronic Communications of the EASST*, vol. 57, 2013.

[24] P. Braun and F. Marschall, *"Transforming Object Oriented Models with BOTL," Electronic Notes in Theoretical Computer Science*, vol. 72, no. 3, pp. 103–117, 2003.

[25] T. Horn, *Bidirectional Model Transformation with FunnyQT*. University of Koblenz-Landau, Germany, 2014.

[26] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, *"Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-update Problem," ACM Transactions on Programming Languages and Systems*, vol. 29, no. 3, 2007.

[27] I. Sasano, Z. Hu, S. Hidaka, K. Inaba, H. Kato, and K. Nakano, *"Toward Bidirectionalization of ATL with GRoundTram,"* in *Theory and Practice of Model Transformations* (J. Cabot and E. Visser, eds.), vol. 6707 of *Lecture Notes in Computer Science*, pp. 138–151, Springer Berlin Heidelberg, 2011.

[28] D. Kolovos, R. Paige, and F. Polack, *"On the evolution of ocl for capturing structural constraints in modelling languages,"* in Rigorous Methods for Software Construction and Analysis, ser. Lecture Notes in Computer Science, J.-R. Abrial and U. Glsser, Eds. Springer Berlin Heidelberg, 2009, vol. 5115, pp. 204–218.

[29] C. Poskitt, M. Dodds, R. F. Paige, and A. Rensink, *"Towards rigorously faking bidirectional model transformations,"* in Proceedings of the Workshop on Analysis of Model Transformations, AMT 2014, Valencia, Spain, ser. CEUR-WS, J. Dingel, J. De Lara, L. L´ucio, and H. Vangheluwe, Eds., vol. 1277. Aachen: RWTH Aachen, Germany, September 2014, pp. 70–75.