ارائه الگوریتم ادغام سه‌طرفه برای مدل‌های یوام‌ال بر اساس مفروضات ذهنی طراحان

محمدرضا شعرباف[1]، دانشجوی کارشناسی ارشد؛ بهمن زمانی[2]، استادیار

۱- دانشکده مهندسی کامپیوتر - دانشگاه اصفهان - اصفهان - ایران - m.sharbaf@eng.ui.ac.ir
۲- دانشکده مهندسی کامپیوتر - دانشگاه اصفهان - اصفهان - ایران - zamani@eng.ui.ac.ir

چکیده: مدل، نمایشی انتزاعی از سیستم نرم‌افزاری است که راهکاری مناسب برای مقابله با پیچیدگی‌های نرم‌افزار می‌باشد. توسعه مدل رانده نیز، با بهره‌گیری از این واقعیت، روشی نوین در توسعه نرم‌افزار است که از مدل به‌عنوان دست‌ساخته اصلی برای ایجاد سیستمی نرم‌افزاری استفاده می‌کند. همین امر موجب شده با افزایش همکاری تیمی، نسخه‌های مختلفی از مدل در طول فرایند توسعه، به‌ویژه در مرحله طراحی، ایجاد شود. برای مدیریت این نسخه‌ها، شناسایی تفاوت‌ها و تطبیق آن‌ها به‌صورت مدلی یکپارچه، ضروری است. درنتیجه، نیاز به ابزار و روش‌هایی برای پشتیبانی از عملیات ادغام نسخه‌های مدل و کاهش مداخله کاربر در آن، موردنیاز است. در این مقاله، الگوریتم و ابزاری برای ادغام سه‌طرفه مدل‌های یوام‌ال ارائه می‌شود که پیش از آغاز ادغام، فرضیات ذهنی هر طراح را دریافت کرده و درنهایت یک مدل ادغامی سازگار ایجاد می‌کند. با توجه به ارزیابی‌های انجام‌شده، عدم وابستگی به کاربر و محیط مدل‌سازی به همراه زمان اجرای کم‌تر نسبت به روش‌های موجود از مزایای روش پیشنهادی می‌باشد.

واژه‌های کلیدی: توسعه مدل رانده، کنترل نسخه، ادغام سه‌طرفه، مدل‌های یوام‌ال.

A Three-way Merging Algorithm for UML Models Based on Developers' Subjective Assumptions

M. Sharbaf[1], MSc Student; B. Zamani[2], Assistant Professor

1- Department of Software Engineering, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran, Email: m.sharbaf@eng.ui.ac.ir
2- Department of Software Engineering, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran, Email: zamani@eng.ui.ac.ir

Abstract: A model is an abstract representation of software system and is an appropriate solution to cope with the complexity of software. Utilizing from this fact, Model-Driven Development is an approach to software development that employs models as main artifacts for building software systems. With the increase in the number of designers in a team, different versions of model arise at any time during the development process, especially in the design phase. To manage these versions it is necessary to be able to identify differences and reconcile them in a single, integrated model. As a consequence, there is a need for tools and techniques to support model merging with less user interference. In this paper, an algorithm and tool for three-way merging of UML models is presented. Before merging, subjective assumptions of each developer is received and at the end, a consistent merged model is provided. According to the evaluations, user and modeling environment independency with less running time than the existing approaches, shows the superiority of this work.

Keywords: Model-driven development, version control, three-way merging, UML models.

تاریخ ارسال مقاله: ۱۳۹۵/۰۴/۲۴
تاریخ اصلاح مقاله: ۱۳۹۵/۰۵/۳۱
تاریخ پذیرش مقاله: ۱۳۹۵/۰۷/۲۲
نام نویسنده مسئول: بهمن زمانی
نشانی نویسنده مسئول: ایران - اصفهان - خیابان هزار جریب - دانشگاه اصفهان - دانشکده مهندسی کامپیوتر.

۱- مقدمه

پیشرفت در توسعه نرم‌افزار منجر به افزایش روزافزون پیچیدگی سیستم‌های نرم‌افزاری شده و تقاضا برای روش‌های جدیدی که طراحان را در مقابله با این امر یاری کنند، افزایش یافته است [۱، ۲]. مدل در مهندسی نرم‌افزار نیز، نمایشی انتزاعی از سیستم نرم‌افزاری است که امکان استدلال و درک بهتر سیستم را برای طراحان فراهم کرده و راهکاری مناسب برای مقابله با پیچیدگی‌های نرم‌افزار می‌باشد. با بهره‌گیری از این واقعیت، توسعه مدل رانده[۱]، یکی از جدیدترین روش‌هایی است که از ایده ارتقاء سطح تجرید در جهت کاهش پیچیدگی‌ها و خودکارسازی تولید کد استفاده می‌کند. در این روش، مدل به‌عنوان دست‌ساخته[۲] اصلی در فرآیند توسعه ایجاد شده و در چرخه تولید نرم‌افزار، دست‌خوش تغییرات عمده‌ای در جهت تولید کد یا سیستم اجرایی قرار می‌گیرد [۳].

امروزه اغلب نرم‌افزارها به‌صورت تیمی، توسعه می‌یابند که موجب می‌شود مدل‌ها علاوه‌بر مزایای یادشده، به‌عنوان واسط ارتباطی مناسب نیز در بین متخصصان استفاده شوند. درنتیجه یک مدل اغلب توسط افراد مختلفی ویرایش شده و این امر نیاز به روش و ابزاری چون سیستم‌های کنترل نسخه[۳] را جهت مدیریت و پشتیبانی از نسخه‌های مختلف مدل به وجود می‌آورد [۴]. سیستم‌های کنترل نسخه با ایجاد محیطی همکارانه امکان ویرایش نسخه‌ای، توسط افراد مختلف را فراهم آورده و موجب بهبود توسعه نرم‌افزار به‌صورت تیمی می‌شوند. اگرچه در سال‌های اخیر توسعه تیمی مدل‌ها گسترش یافته‌است، با این‌حال اکثر سیستم‌های کنترل نسخه فعلی تنها از توسعه نرم‌افزار کد محور پشتیبانی می‌کنند و به توسعه تیمی نرم‌افزار به شیوه مدل محور پرداخته نشده است.

سیستم‌های کنترل نسخه دارای دو استراتژی نسخه‌بندی بدبینانه[۴] و خوش‌بینانه[۵] برای پشتیبانی از تکامل همزمان یک دست‌ساخته می‌باشند [۵]. در استراتژی خوش‌بینانه که اکثر سیستم‌های کنترل نسخه از آن استفاده می‌کنند، امکان اعمال تغییرات به‌طور همزمان توسط کاربران مختلف وجود داشته ولی در پایان بایستی نسخه‌های مختلف ادغام گردند. ادغام سه‌طرفه[۶]، یکی از محبوب‌ترین تکنیک‌های ادغام است که بر ترکیب دو نسخه از یک دست‌ساخته با توجه به نسخه اصلی تأکید دارد [۶]. این تکنیک با درنظرگیری نسخه اصلی دست‌ساخته، بستر لازم جهت تشخیص تداخل[۷] را فراهم می‌کند. بااین‌حال عملیات ادغام در نسخه‌بندی مدل با چالش‌های عمده‌ای مواجه است که از مهم‌ترین آن‌ها می‌توان به ضعف در تشخیص کامل تغییرات و وقوع ناسازگاری و تداخل در نسخه ادغامی اشاره نمود [۷].

در سال‌های اخیر الگوریتم‌های متعددی به منظور رفع این چالش‌ها ارائه شده است که برخی جهت تشخیص عناصر معادل، نیازمند وجود شناسه یکتا برای عناصر مدل می‌باشند و برخی دیگر دنباله تغییرات اعمال‌شده را دریافت می‌کنند [۵]. بااین‌حال بسیاری از ابزارهای مدل‌سازی از ثبت دنباله تغییرات، پشتیبانی نمی‌کنند و داشتن شناسه یکتا نیز بدون اطلاع از مفروضات ذهنی طراح، تضمینی بر معادل بودن دو عنصر با نام‌های متفاوت نمی‌باشد.

به‌همین‌منظور در این مقاله الگوریتمی به شیوه ادغام سه‌طرفه ارائه خواهد شد که علاوه‌بر انجام عملیات مقایسه، مفروضات ذهنی طراحان را نیز در محاسبه شباهت‌ها لحاظ نموده و با استفاده از لیست تشابهات اقدام به ایجاد مدل ادغام‌شده و سازگار از نسخه‌های موجود می‌نماید. الگوریتم ارائه‌شده قابل استفاده برای تمامی مدل‌های یوام‌ال می‌باشد و با تغییرات جزئی از سایر زبان‌های مدل‌سازی نیز پشتیبانی می‌کند. اگرچه در این مقاله از نمودار کلاس جهت بیان روش موردنظر استفاده شده است.

در ادامه مقاله، ابتدا مفاهیم توسعه مدل‌رانده و سیستم‌های کنترل نسخه در بخش‌های ۲ و ۳ معرفی شده و پس از آن روش‌های ادغام، شیوه‌های مقایسه، مسئله موردنظر و محک Project_Management در بخش ۴ بررسی می‌گردد. کارهای مرتبط در بخش ۵ بررسی شده و الگوریتم ادغام سه‌طرفه پیشنهادی در دو فاز مقایسه و ادغام به همراه پیاده‌سازی انجام‌شده در بخش ۶ ارائه خواهد شد. بخش ۷، به ارزیابی الگوریتم پیشنهادی و مقایسه آن با دو الگوریتم پیاده‌سازی شده می‌پردازد و درنهایت مقاله با نتیجه‌گیری در بخش ۸ به پایان می‌رسد.

۲- توسعه مدل رانده

سابقه ارتقای سطح تجرید به معرفی زبان اسمبلی به‌عنوان تجریدی بر روی کد ماشین باز می‌گردد. پس از آن، زبان‌های نسل سوم مطرح شدند که در آن برخی از وظایف سطح پایین را به‌جای برنامه‌نویس به کامپایلر محول می‌کردند. زبان‌های شئ‌گرا نیز، تجریدهای بالاتری مانند نوع داده انتزاعی[۸] را مطرح کردند [۸]. توسعه مدل رانده این رویه را ادامه داد و با معرفی تجریدی بالاتر از کد، یعنی مدل، آن را در سطوح مختلف چرخه حیات نرم‌افزار گسترش داد.

در توسعه مدل رانده، مدل به‌عنوان دست‌ساخته اصلی در فرآیند توسعه ایجاد شده و در چرخه تولید نرم‌افزار قرار می‌گیرد. به‌طوری‌که با به‌کارگیری مجموعه روش‌هایی تحت عنوان تبدیل مدل، سیستم اجرایی یا کد پیاده‌سازی به‌صورت (نیمه) خودکار از آن تولید می‌شود [۳]. در دهه ۸۰، فناوری شئ‌گرا با اصل «هر چیز یک شئ است» بیان شد، اما در توسعه مدل رانده اصل مهم با عبارت «هر چیز یک مدل است» بیان می‌شود [۹]. می‌توان گفت در مهندسی مدل‌رانده، مدل انتزاعی از دنیای واقع و فراتر از طرح و نقشه ساده از سیستم است که به‌عنوان عنصر محوری در فرآیند مهندسی نرم‌افزار برای دست‌یابی به سیستم هدف در نظر گرفته می‌شود. درنتیجه طراحان می‌توانند مدلی مستقل از تکنیک‌های پیاده‌سازی و نزدیک‌تر به زمینه مسئله ایجاد نموده و به این ترتیب از پیچیدگی توسعه سیستم‌های نرم‌افزاری بکاهند. این امر امکان استدلال و درک بهتر سیستم را برای طراحان فراهم می‌آورد.

هر مدل بایستی منطبق بر یک زبان مدل‌سازی باشد، که در اصطلاح به آن فرامدل[۹] می‌گوییم. زبان مدل‌سازی، زبانی به‌منظور بیان

اطلاعات یا ویژگی‌های سیستم‌ها در ساختاری مشخص، با استفاده از مجموعه‌ای از قوانین پایدار می‌باشد. زبان مدل‌سازی یکپارچه[۱۰]، یوام‌ال، نمونه‌ای از یک زبان مدل‌سازی بصری برای حمایت از طراحی و توسعه سیستم‌های شئ‌گرا است. این زبان که در حال‌حاضر به‌عنوان استاندارد مدل‌سازی نرم‌افزار پذیرفته شده است، نمودارهایی ارائه می‌دهد که به دو دسته نمودارهای رفتاری و ساختاری تقسیم می‌شوند. نمودارهای رفتاری جنبه پویای سیستم را بیان کرده و نمودارهای ساختاری جنبه ایستای آن را به تصویر می‌کشند. نمودار کلاس که در این پژوهش استفاده شده، از نمودارهای ساختاری است.

۳- سیستم کنترل نسخه

در طول چرخه توسعه نرم‌افزار، بسیاری از فعالیت‌ها بایستی به‌طور مداوم و با همکاری تیمی از متخصصان انجام شود. خروجی این فعالیت‌ها نسخه‌های جدید یا ویرایش شده‌ای هستند که تنها در صورت مدیریت صحیح منجر به موفقیت در فرآیند توسعه نرم‌افزار می‌شوند. سیستم‌های کنترل نسخه، یکی از مهم‌ترین ابزارهایی هستند که ویژگی‌های موردنیاز جهت مدیریت نسخه‌های مختلف ایجاد شده در طول فرآیند توسعه یک سیستم نرم‌افزاری پیچیده را فراهم می‌کنند. این سیستم‌ها به‌طورکلی سه هدف را دنبال می‌کنند [۱۰]:

۱) ذخیره تاریخچه تکامل دست‌ساخته‌های نرم‌افزار،
۲) پشتیبانی توسعه همزمان توسط چند کاربر،
۳) مدیریت شاخه‌های مختلف توسعه

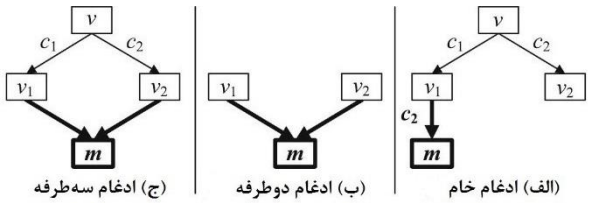
سیستم نسخه‌بندی شامل مجموعه‌ای از روش‌ها برای پشتیبانی از ذخیره دست‌ساخته‌های نرم‌افزاری بوده و با ایجاد محیطی همکارانه امکان ویرایش نسخه‌ای، توسط افراد مختلف را فراهم می‌آورد. این سیستم‌ها دارای دو استراتژی نسخه‌بندی بدبینانه و خوش‌بینانه برای پشتیبانی از تکامل همزمان یک دست‌ساخته می‌باشد [۵]. در مواردی که از نسخه‌بندی بدبینانه استفاده می‌شود، دست‌ساخته‌ای در زمان تغییر توسط یک کاربر قفل شده و تا زمان آزادسازی آن، کاربر دیگری امکان دریافت و اعمال تغییرات در دست‌ساخته موردنظر را ندارد. گرچه این روش از تداخل جلوگیری می‌کند ولی امکان وقوع زمان بیکاری وجود دارد. به‌منظور جلوگیری از وقوع این زمان بیکاری، نسخه‌بندی خوش‌بینانه ارائه شد. در این استراتژی که حیطه‌کاری مقاله حاضر است، دو کاربر همزمان قادر به دریافت و اعمال تغییرات بر نسخه‌ای از سیستم می‌باشند اما در زمان تحویل نسخه‌ها به سیستم، ابتدا بایستی فرآیند ادغام نسخه‌های ویرایش‌شده، انجام شود. فرآیند ادغام می‌تواند شامل یکپارچه‌سازی نسخه نهایی، تشخیص ناسازگاری و کشف و رفع تداخل باشد.

۴- ادغام مدل

به‌منظور یکپارچه‌سازی و ایجاد نسخه‌ای جامع از نسخه‌های ایجادشده از یک مدل نیاز به استفاده از فرآیندی برای ادغام می‌باشد. به‌طورکلی این فرآیند در چهار فاز مقایسه، بررسی انطباق، ادغام و بازسازی انجام می‌شود [۱۱]. در فاز مقایسه، عناصر موجود در نسخه‌های مختلف با یکدیگر مقایسه شده و لیستی از عناصر معادل استخراج می‌گردد. لیست تشابه به‌دست‌آمده در این مرحله، در فاز دوم جهت کشف و رفع تداخل‌های ممکن بررسی و اصلاحات موردنیاز انجام می‌شود. در فاز سوم با توجه به ویژگی‌های عناصر معادل و روش ادغام، عناصر نسخه ادغامی ایجاد شده و در فاز بازسازی، این عناصر به‌منظور حفظ سازگاری مدل نهایی، موردبررسی قرار خواهند گرفت. با توجه به فازهای بیان‌شده، دو شیوه مقایسه و سه روش ادغام مطرح شده‌اند که هر یک به نحوی سعی در پوشش فرآیند ادغام دارند.

۴-۱- روش‌های ادغام

یکپارچه‌سازی نسخه‌های مختلف یک دست‌ساخته نیازمند استفاده از روشی برای ادغام است. همان‌گونه که در شکل ۱ نمایش داده شده است، سه روش ادغام خام[۱۱]، ادغام دوطرفه[۱۲] و ادغام سه‌طرفه وجود دارد [۵]. در ادامه هر یک به‌طور اجمالی بیان می‌شود.



شکل ۱: روش‌های ادغام [۱۲]

۴-۱-۱- ادغام خام

در ادغام خام، به ترتیب دنباله‌ای از تغییرات ایجادشده توسط طراح اول و دوم، به سادگی بر روی نسخه اصلی اعمال شده تا نسخه ادغامی ایجاد شود. در طول این نوع ادغام درصورتی‌که هر یک از تغییرات نتواند اعمال شود (به‌طورمثال، به علت اعمال بر عنصری ناموجود)، ناسازگاری رخ داده است. در این روش امکان تشخیص تغییرات موازی که زمینه‌ساز تداخل و ناسازگاری می‌باشند، وجود ندارد. شکل ۱- الف ادغام خام را نشان می‌دهد که در آن v نسخه اصلی دست‌ساخته و v_1 و v_2 به ترتیب نسخه‌هایی هستند که با تغییرات c_1 و c_2 توسط طراح اول و دوم به وجود آمده‌اند؛ m نیز نسخه نهایی (ادغامی) می‌باشد.

۴-۱-۲- ادغام دوطرفه

همان‌گونه که در شکل ۱- ب نشان داده شده است، در ادغام دوطرفه، دو نسخه ویرایش‌شده v_1 و v_2 مقایسه شده و بدون توجه به نسخه اصلی، با یکدیگر اجماع شده تا نسخه ادغامی m را ایجاد کنند. در این روش امکان تشخیص حذف یا اضافه شدن عناصر وجود ندارد، زیرا در زمان ادغام، نسخه اصلی در نظر گرفته نمی‌شود. بنابراین، این روش نیز همانند ادغام خام، جهت تشخیص تغییرات و کشف تداخل مناسب نمی‌باشد.

۴-۱-۳- ادغام سه‌طرفه

با توجه به شکل ۱- ج، در ادغام سه‌طرفه، علاوه‌بر دو نسخه ویرایش‌شده v_1 و v_2، نسخه اصلی v نیز در نظر گرفته می‌شود. در این روش برای تشخیص دنباله تغییرات، مقایسه‌ای میان v و v_1 و مقایسه دیگری میان v و v_2 انجام شده و عملیات ادغام با اجتماع تغییرات دو دنباله انجام می‌شود. با استفاده از ادغام سه‌طرفه امکان تشخیص اضافه، حذف و به‌روز شدن عناصر در نسخه ویراش‌شده وجود داشته که این امر شرایط لازم جهت کشف و رفع تداخل و جلوگیری از ناسازگاری در زمان ادغام را فراهم می‌نماید. همین امر موجب شده که در این مقاله از روش ادغام سه‌طرفه جهت انجام فرآیند ادغام استفاده شود.

۴-۲- شیوه مقایسه

شیوه مقایسه در ادغام مدل را می‌توان به دو دسته کلی تقسیم کرد [۵]. دسته اول شیوه‌ای که مبتنی بر حالت است و بر اساس حالت ثابتی از نسخه‌های مختلف مدل، عمل مقایسه را انجام می‌دهد و دسته دوم شیوه‌ای که مبتنی بر عملیات است و دنباله‌ی پویایی از تغییرات اعمال شده توسط طراح را آنالیز می‌نماید. هر دو نوع ادغام دوطرفه و سه‌طرفه قابل بیان با شیوه مبتنی بر حالت و مبتنی عملیات می‌باشند ولی نوع ادغام خام تنها به کمک شیوه مبتنی بر عملیات قابل بیان است [۱۳].

۴-۲-۱- شیوه مبتنی بر حالت

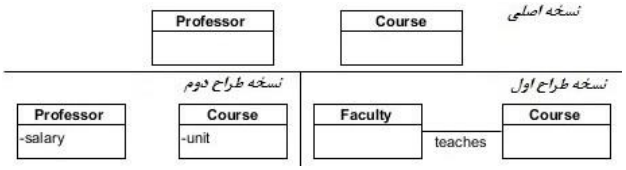
اگر شیوه مقایسه در ادغام تنها مبتنی بر مقایسه حالت‌های نهایی عناصر موجود در نسخه ویراش‌شده باشد، به آن مقایسه مبتنی بر حالت گفته می‌شود. در این شیوه از هیچ اطلاعات اضافه‌ای، ازجمله اطلاعات مربوط به تکامل نسخه دست‌ساخته، استفاده نمی‌شود و این موجب شده همیشه امکان تشخیص تمامی تغییرات وجود نداشته باشد. بااین‌حال، عدم وابستگی به ابزار مدل‌سازی از مزایای اصلی این شیوه است.

۴-۲-۲- شیوه مبتنی بر عملیات

در شیوه مبتنی بر عملیات، بایستی تاریخچه‌ای از تغییرات اعمال شده توسط طراحان، ذخیره گردد. پس از آن با مقایسه، مرتب‌سازی، حذف و تغییر برخی تغییرات، دنباله‌ای از آن‌ها ایجاد شده که به کمک آن می‌توان نسخه نهایی را ایجاد نمود. از مزایای این شیوه امکان تشخیص کامل تغییرات است که شرایط لازم جهت کشف تداخل را فراهم می‌سازد. بااین‌حال، تنها برخی از ابزارهای مدل‌سازی امکان ذخیره تغییرات ایجادشده توسط طراحان را فراهم می‌کنند که روش ذخیره‌سازی آن‌ها نیز یکسان نبوده و موجب وابستگی به ابزار مدل‌سازی می‌شود.

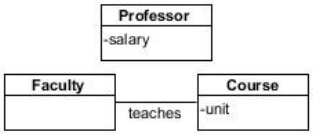
۴-۳- بحث و بررسی

اگرچه از سه روش ادغام موجود که در بخش ۴-۱ مطرح شد، روش ادغام سه‌طرفه مناسب‌ترین روش جهت برطرف کردن چالش‌های موجود در ادغام مدل‌ها می‌باشد [۵]، بااین‌حال اکثر الگوریتم‌های ادغام کنونی توانایی لازم جهت تشخیص کامل تغییرات و تأثیری که آن‌ها در زمان ادغام بر سایر عناصر دارند را نداشته و نیازمند بهره‌گیری از نظر کاربر در زمان ادغام هستند [۱۴]. در این شرایط توجه به این موضوع اهمیت داشته که کاربرِ مسئول ادغام نسخه‌ها، ممکن است از ذهنیت و استدلال طراحان هر دو نسخه ویرایش‌شده آگاهی کافی نداشته باشد و با اتخاذ تصمیم نادرست منجر به نقص در نسخه ادغام شده، گردد. به‌طورمثال، شکل ۲ قسمتی از نمودار کلاس یک سیستم آموزشی را نشان می‌دهد که توسط دو طراح ویرایش شده است. نسخه اصلی این مدل شامل دو کلاس Professor و Course بوده است. طراح اول با دریافت نسخه اصلی، نام کلاس Professor را به Faculty تغییر داده و رابطه teaches را میان دو کلاس Faculty و Course برقرار کرده است. طراح دوم نیز، خصیصه salary را به کلاس Professor و unit را به کلاس Course اضافه کرده است.



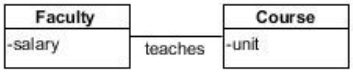
شکل ۲: نمونه‌ای از نسخه‌های نمودار کلاس یک سیستم آموزشی

با توجه به تغییرات ایجادشده توسط دو طراح، فرآیند ادغام سه‌طرفه با تشخیص شباهت کلاس Course در سه نسخه و کلاس Professor در دو نسخه اصلی و طراح دوم آغاز شده و کلاس Faculty و رابطه teaches را به‌عنوان عناصر جدیدی که به مدل اضافه شده‌اند، در نظر می‌گیرد. این امر موجب شده خروجی حاصل از ادغام به‌صورت شکل ۳، باشد.



شکل ۳: نسخه خروجی ادغام بدون آگاهی از ذهنیت طراحان

درحالی‌که کلاس Faculty در نسخه طراح دوم، همان کلاس Professor بوده و بایستی خروجی حاصل از ادغام به‌صورت شکل ۴ باشد. روش پیشنهادی، به‌منظور رفع این مشکل ارائه می‌شود.



شکل ۴: نسخه خروجی ادغام با آگاهی از ذهنیت طراحان

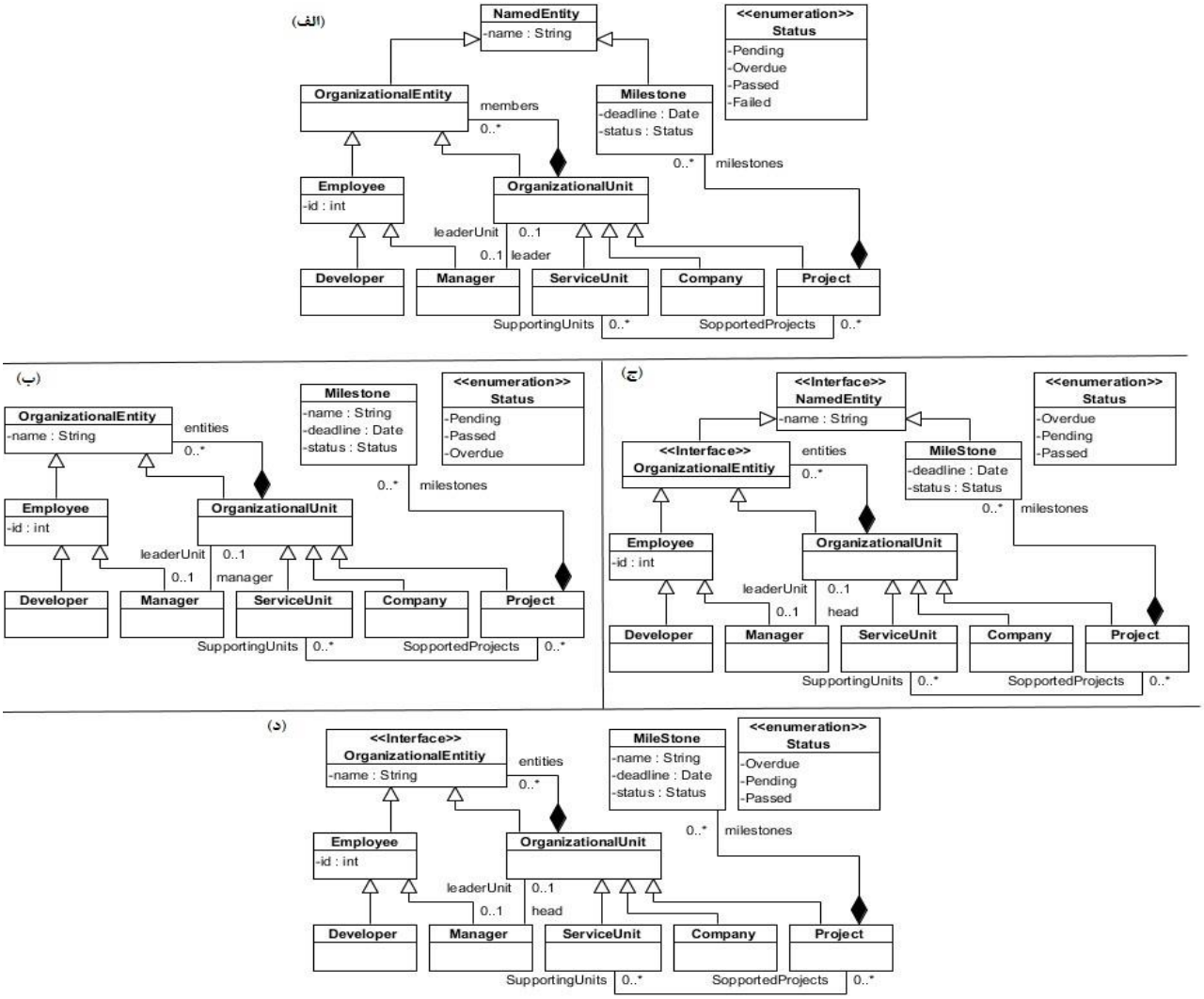
۴-۴- محک Project_Management

به‌منظور ارزیابی و مقایسه روش پیشنهادی و ابزار ایجادشده، از مثال Project_Management که در شکل ۵ آمده است، استفاده می‌شود. این مثال که در [۱۵] مطرح شده است، نمودار کلاس یک سیستم مدیریت پروژه می‌باشد. نسخه اصلی این نمودار که در قسمت بالایی شکل ۵ نمایش داده شده است، چگونگی تقسیم‌بندی موجودیت‌های نام‌دار سیستم مدیریت پروژه و روابط بین آن‌ها را بیان نموده است. این نسخه توسط دو طراح ویرایش شده و نسخه‌های سمت چپ و سمت راست ایجاد شده است. به‌طورکلی تغییرات را می‌توان به سه دسته (الف) تغییرات مشترک، (ب) تغییرات نامتناقض و (ج) تغییرات متناقض تقسیم نمود. بر اساس این تقسیم‌بندی تغییرات اعمال‌شده توسط این دو طراح در جدول ۱ آمده است.

سازگاری نسخه حاصل از ادغام با فرامدل نمودار کلاس یوام‌ال و میزان شباهت آن با نسخه خروجی مورد انتظار از ادغام که در قسمت پایینی شکل ۵ نشان داده شده است، از معیارهای سنجش صحت این محک می‌باشند. از دیگر معیارها می‌توان به توانایی الگوریتم در پشتیبانی از سه دسته تغییر معرفی‌شده در جدول ۱ اشاره نمود.

جدول ۱: دسته‌بندی تغییرات دو طراح در ویرایش نسخه اصلی

دسته	تغییر
تغییرات مشترک	- تغییر نام ارجاع members از کلاس OrganizationalUnit به entities. - حذف لیترال Failed از کلاس شمارشی Status.
تغییرات نامتناقض	- افزودن خصیصه name به هر دو کلاس Milestone و OrganizationalEntity در نسخه سمت چپ - تغییر نام کلاس Milestone به MileStone در نسخه سمت راست
تغییرات متناقض	- حذف کلاس NamedEntity در نسخه سمت چپ و تبدیل آن به یک کلاس واسط - تغییر نام ارجاع leader از کلاس OrganizationalUnit به manager در نسخه سمت چپ و به head در نسخه سمت راست



شکل ۵: نمودار کلاس مربوط به محک Project_Management شامل نسخه اصلی (الف)، نسخه طراح اول (ب)، نسخه طراح دوم (ج) و نسخه مورد انتظار از ادغام (د) [۱۵]

۵- کارهای مرتبط

سابقه پژوهش در ادغام مدل‌ها به دهه ۱۹۹۰ میلادی باز می‌گردد. جایی‌که با دسته‌بندی شیوه‌های مقایسه به دو شیوه مبتنی بر حالت و مبتنی بر عملیات و تفکیک روش‌های ادغام به سه روش ادغام خام، دوطرفه و سه‌طرفه [۱۲]، زمینه برای پژوهش‌های بعدی فراهم شده و کارهای مختلفی در جهت مقایسه و ادغام مدل‌ها ارائه شد.

از اولین کارها، الگوریتمی مبتنی بر عملیات برای محاسبه و ادغام سه‌طرفه تغییرات پایه با توجه به نظرات کاربران بود که تنها برای مدل‌های ساده یوام‌ال قابل استفاده می‌باشد [۱۶]. زبان‌های مقایسه و ادغام اپسیلون[۱۳] [۱۱]، کار دیگری است که امکان تعیین معیارهای مقایسه به شیوه مبتنی بر حالت و ادغام عناصر مشابه به روش دوطرفه را فراهم کرده و قابل استفاده برای تمامی مدل‌ها می‌باشد. همچنین مواردی دیگری از این قبیل وجود دارند که هدف تمامی آن‌ها، تنها انجام عملیات ادغام برای مدل‌ها بوده و علی‌رغم اهمیت کشف و رفع تداخل‌ها، هیچکدام به این موضوع نپرداخته‌اند.

بر همین اساس، اکثر پژوهش‌های جدید با تمرکز بر تداخل‌ها و ناسازگاری‌ها، اقدام به ارائه راهکاری برای کشف و رفع آن‌ها نموده‌اند. این پژوهش‌ها را که اکثراً از روش ادغام سه‌طرفه استفاده می‌کنند، می‌توان به دو دسته کلی تقسیم نمود. دسته اول راهکارهایی چون [۱۷، ۱۸] هستند که پس از انجام مقایسه، اصلاح ناسازگاری‌ها را به‌طور کامل به کاربر واگذار می‌کنند یا مانند [۱۹]، با ارائه راهنمایی‌ها و پیشنهادات ممکن، کاربر را در اصلاح آن‌ها کمک می‌نمایند. لازم به ذکر است که تمامی این راهکارها نیاز به تعامل با کاربر در زمان اجرای ادغام دارند. دسته دوم راهکارهایی چون [۱۹، ۲۰] می‌باشند که پیش از انجام ادغام، با اولویت دادن به مجموعه‌ای از تغییرات یا تعریف سیاست‌های ادغام، امکان اجرای نیمه‌خودکار فرآیند را فراهم می‌کنند. بااین‌حال، این دسته از راهکارها برای کشف و اصلاح تداخل‌های ساختاری مناسب بوده و در مواردی مانند تغییر بیان‌شده در شکل ۲ که نیازمند درک مفاهیم معنایی می‌باشد، کارایی لازم را ندارد.

همچنین راهکارهای مبتنی بر حالت برای ادغام مدل‌ها، نیازمند تعیین شباهت‌ها و تفاوت‌های نسخه‌های مختلف می‌باشند که به همین منظور اکثراً روشی برای مقایسه مدل‌ها ارائه کرده‌اند. اگرچه برخی از آن‌ها مانند [۲۱] از روش‌ها و ابزارهای موجود برای مقایسه استفاده می‌کنند. خروجی فاز مقایسه، لیست عناصر مشابه [۱۴] یا مدلی از شباهت‌ها [۲۱] یا تفاوت‌های [۲۲] دو نسخه می‌باشد که در زمان ادغام استفاده می‌شود.

از راهکارهای موجود برخی مانند [۱۷، ۱۹، ۲۰] وابسته به زبان‌های مدل‌سازی خاص می‌باشند ولی برخی دیگر مانند [۲۱، ۲۳] علی‌رغم ارائه با استفاده از نمونه‌ای از زبان مدل‌سازی خاص، مستقل از فرامدل بوده و قابلیت ادغام مدل‌های سایر زبان‌های مدل‌سازی را نیز پشتیبانی می‌کنند. بنابراین با توجه به اینکه راهکار پیشنهادی نیز با دریافت فرامدل‌های مختلف امکان ادغام مدل‌های ایجادشده در زبان‌های مدل‌سازی مختلف را فراهم می‌کند، در ادامه دو الگوریتم ارائه‌شده در این دسته که به‌صورت افزونه‌ای تحت اکلیپس پیاده‌سازی شده‌اند، بررسی شده تا با راهکار پیشنهادی پژوهش حاضر مقایسه شوند. این دو الگوریتم عبارت‌اند از:

- **الگوریتم اول:** در [۲۱] روشی صوری برای ادغام سه‌طرفه مدل‌ها ارائه شده است. این روش که اولین بار برای مدل‌های تحت EMF[۱۴] بررسی شده است، مستقل از فرامدل بوده و از مدل‌های یوام‌ال نیز پشتیبانی می‌کند. اگرچه امکان انجام فاز مقایسه در این الگوریتم فراهم نشده و شناسایی تفاوت‌ها با استفاده از ابزارهای موجود بایستی انجام شود، ولی برای تعیین شباهت‌ها نیازی به شناسه یکتا ندارد. به‌منظور انجام ادغام دو دسته قانون تعریف شده است. دسته اول قانون‌های مستقل از زمینه[۱۵] برای عناصری که بایستی بدون در نظر گرفتن زمینه به نسخه ادغامی منتقل شوند و دسته دوم قانون‌های حساس به زمینه[۱۶] هستند که عناصری که بایستی سازگاری آن‌ها بررسی شود را به نسخه ادغامی اضافه می‌نماید. در این حالت، در صورت عدم سازگاری، تصمیم‌گیری و اصلاح عناصر به کاربر واگذار می‌گردد. مرتبه زمانی این الگوریتم که پیاده‌سازی آن به‌صورت افزونه‌ای تحت اکلیپس می‌باشد، در بدترین حالت $O(n^3)$ می‌باشد که n تعداد عناصر مدل است.
- **الگوریتم دوم:** در [۲۳]، راهکاری بر اساس مفاهیم گراف ارائه شده است که در آن درج و حذف عناصر از نسخه اصلی مدل، به‌صورت تغییراتی در گراف در نظر گرفته شده و در دو فاز انجام می‌شود. در فاز اول درج عناصر نسبت به حذف آن‌ها اولویت داشته و دنباله‌ای از تغییرات بر نسخه اصلی مدل اعمال شده تا نسخه ادغامی موقتی ایجاد شود. پس از آن در فاز دوم، با بررسی حالت‌های عناصر موجود در نسخه ایجادشده، سازگاری آن‌ها بررسی‌شده و نسخه ادغامی نهایی ایجاد می‌شود. این راهکار به‌صورت افزونه‌ای تحت اکلیپس برای مدل‌های یوام‌ال پیاده‌سازی شده و از هر دو روش مقایسه مبتنی بر حالت و مبتنی بر عملیات پشتیبانی می‌کند. معیار شباهت در این روش شناسه یکتا بوده و در فاز دوم، نیاز به دریافت بازخورد کاربر دارد. مرتبه زمانی این الگوریتم $O(n^2) + O(m)$ است که m تعداد تغییرات و n تعداد عناصر نسخه ادغامی موقت است. اگر تعداد تغییرات را مقدار کوچک فرض کنیم، می‌توان گفت مرتبه زمان آن در بدترین حالت $O(n^2)$ می‌باشد.

اگرچه تمامی راهکارهای موجود سعی بر پشتیبانی از کشف و رفع تداخل‌ها از طریق تعامل با کاربر در زمان اجرای ادغام یا تعریف قواعدی برای خودکارسازی و کاهش نیاز به حضور وی داشته‌اند، بااین‌حال هیچ‌یک از این راهکارها، ایده دریافت مفروضات ذهنی طراح پیش از ادغام و عدم‌حضور وی در زمان اجرای ادغام را مطرح نکرده‌اند.

در راهکار پیشنهادی در این پژوهش از این ایده برای کشف و رفع تداخل‌ها استفاده شده است.

۶- الگوریتم ادغام سه‌طرفه پیشنهادی

همان‌گونه که در بخش ۴-۳ بررسی شد، عدم آگاهی از ذهنیت طراحان می‌تواند منجر به ایجاد مدلی دور از انتظار در زمان ادغام گردد. به‌منظور رفع این مشکل بایستی هر دو طراح در زمان ادغام حاضر شده و در هر مرحله از ادغام اطلاعات موردنیاز از آن‌ها دریافت شود، که این امر تا حدی دشوار بوده و نیاز به صرف هزینه اضافه است.

در مقاله حاضر پیشنهاد می‌شود، بخشی از اطلاعات موردنیاز که پیش از آغاز ادغام قابل دریافت می‌باشند را از طراحان دریافت کرده و در زمان ادغام با توجه به اطلاعات دریافتی تصمیم مناسب اتخاذ گردد تا علاوه‌بر خودکارسازی ادغام، نیازی به حضور هیچ‌یک از طراحان در زمان ادغام نباشد. به همین منظور الگوریتمی شامل دو فاز کلی، مقایسه و ادغام ارائه می‌شود. خروجی فاز مقایسه که در زمان تحویل نسخه ویرایش‌شده برای هر یک از طراحان اجرا شده و در صورت نیاز اطلاعات لازم را از آن‌ها دریافت می‌کند، لیستی از عناصر مشابه در نسخه اصلی و نسخه ویرایش شده است که در زمان ادغام استفاده می‌شود. همچنین فاز دوم شامل سه گام استنتاج، ادغام و بازسازی نسخه خروجی است. در گام اول با توجه به نسخه‌های ورودی و دو لیست تشابه به‌دست‌آمده از فاز مقایسه، عملیات آماده‌سازی نسخه‌ها برای ادغام انجام می‌شود. در گام دوم عناصر خروجی که بایستی در نسخه ادغام وجود داشته باشند محاسبه شده و خصوصیات هر یک با توجه به دو نسخه‌ی ویرایشی تنظیم می‌شود. در گام آخر نیز، سازگاری عناصر ادغامی با یکدیگر بررسی می‌شود. این الگوریتم قابل استفاده برای تمامی مدل‌های یوام‌ال می‌باشد و با تغییرات جزئی از سایر زبان‌های مدل‌سازی نیز پشتیبانی می‌کند. در ادامه این قسمت جزئیات مربوط به هر یک بررسی خواهد شد.

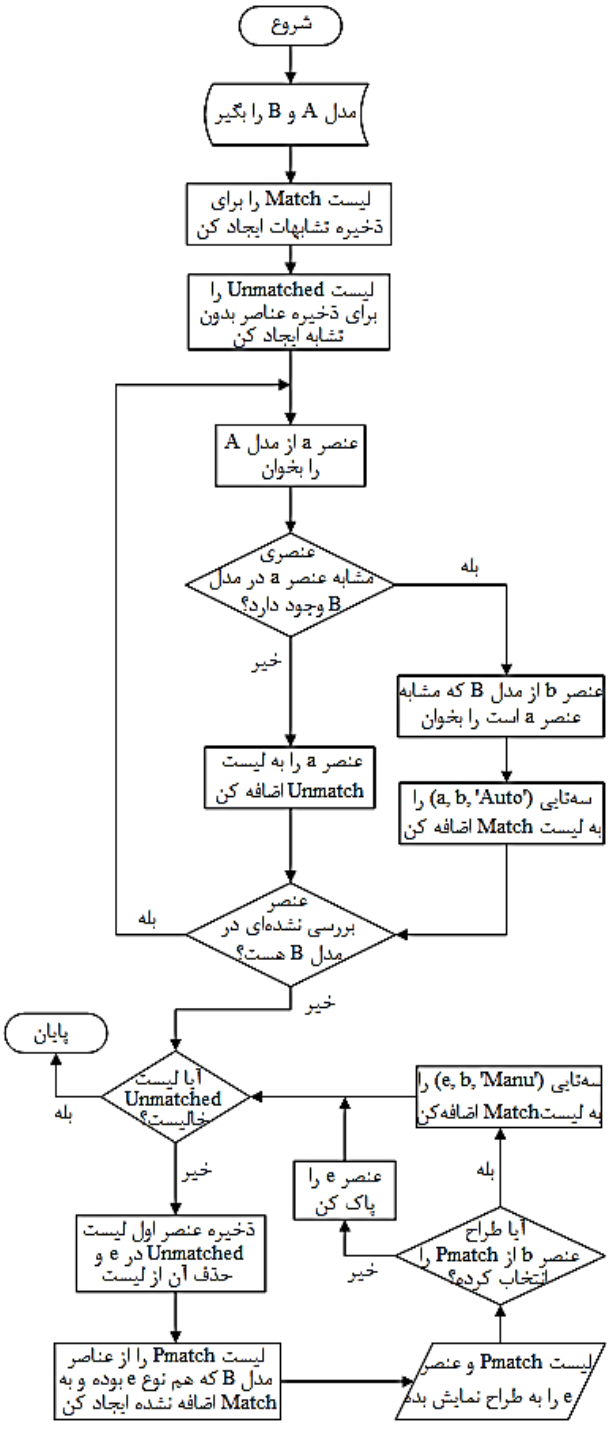
۶-۱- فاز مقایسه

اولین گام در ادغام، تعیین عناصر مشابه و تشخیص تغییرات ایجادشده در نسخه‌ها می‌باشد. از دو شیوه کلی مقایسه که در بخش ۴-۲ معرفی شد، شیوه مبتنی بر عملیات، وابسته به محیط و ابزار مدل‌سازی بوده و تمامی ابزارهای موجود از آن پشتیبانی نمی‌کنند. به همین دلیل در الگوریتم پیشنهادی مقایسه که در

شکل ۶ نمایش داده شده است، شیوه مبتنی بر حالت انتخاب شده و برای بهبود نتایج حاصل از مقایسه، امکان تعامل با کاربر نیز به آن افزوده شده است.

در این الگوریتم ابتدا دو مدل (به‌طورمثال نسخه اصلی و نسخه طراح اول) به‌عنوان ورودی دریافت شده و لیستی تحت عنوان Match برای نگهداری تشابهات ایجاد می‌شود. هر تشابه نمونه‌ای از ساختاری سه‌تایی است که به ترتیب شامل عنصری از مدل اول، عنصری از مدل دوم و پارامتری برای تعیین وضعیت می‌باشد. این پارامتر دارای یکی از دو مقدار 'Auto' یا 'Manu' بوده و سه‌تایی‌هایی که به لیست تشابه اضافه می‌شوند به یکی از دو حالت زیر خواهند بود:

- حالت (a, b, 'Auto') که بیان می‌کند عنصر a از مدل اول با عنصر b از مدل دوم مشابه بوده و تشابه به‌صورت خودکار تشخیص داده شده است.
- حالت (a, b, 'Manu') که بیان می‌کند عنصر a از مدل اول با عنصر b از مدل دوم مشابه بوده و تشابه به‌صورت دستی توسط کاربر تعیین شده است.



شکل ۶: الگوریتم پیشنهادی برای فاز مقایسه

به‌منظور ایجاد لیست تشابهات ابتدا هر عنصر از مدل اول با تک تک عناصر مدل دوم مقایسه شده و در صورت تشابه، یک سه‌تایی با وضعیت 'Auto' به لیست اضافه می‌شود. ازآنجایی‌که رابطه بین عناصر مدل اول و عناصر مدل دوم یک‌به‌یک است، با یافتن عنصر مشابه برای عنصری در مدل اول، عملیات جستجو برای آن عنصر به اتمام رسیده و عنصر مشابه آن در مدل دوم نیز دیگر بررسی نمی‌شود. درصورتی‌که هیچ عنصر مشابهی برای عنصر مدل اول یافت نشود، آن را به لیست Unmatched که عناصر بدون مشابه از مدل اول را نگه‌داری می‌کند فرستاده تا در مرحله بعد توسط کاربر بررسی شود.

با توجه به این موضوع که نام‌ها و روابط در نمودارهای یوام‌ال بیانگر مفاهیم می‌باشند [۲۴]، معیار اصلی شباهت در تعیین دو عنصر مشابه، به ترتیب برابری نوع و نام آن دو عنصر، تشابه فضای نام و برابری سایر مشخصات آن‌ها است. این مشخصات برای هر عنصر در فرامدل آن تعیین شده است. به‌عنوان نمونه، اگر عنصر موردنظر یک کلاس باشد، ازجمله مشخصه‌های آن می‌توان به برابری وضعیت Abstract بودن آن اشاره کرد و درصورتی‌که عنصر موردنظر رابطه‌ای دوطرفه بین دو کلاس باشد، تشابه این دو کلاس اهمیت دارد.

پس از پایان بررسی تک‌تک عناصر مدل اول، درصورتی‌که عنصری وجود داشته باشد که به‌صورت خودکار، مشابهی برای آن یافت نشده باشد، آن عنصر به همراه لیستی از عناصر هم‌نوع خود از مدل دوم که برای آن‌ها نیز مشابهی یافت نشده است، به طراح نمایش داده شده تا از ذهنیت وی آگاه شویم. در صورت انتخاب عنصری از مدل دوم توسط طراح، آن عنصر به همراه عنصر مدل اول با وضعیت 'Manu' به‌صورت یک سه‌تایی به لیست تشابه اضافه می‌شوند. در پایان این الگوریتم لیست تشابهات دو مدل ورودی، به خروجی ارسال می‌شود. عدم‌نیاز به شناسه یکتا در انجام مقایسه، مستقل از محیط و ابزار مدل‌سازی بودن و فراهم‌سازی امکان دریافت مفروضات ذهنی طراح در زمان تحویل نسخه ویرایش‌شده به سیستم کنترل نسخه، از مزایای اصلی الگوریتم مقایسه ارائه‌شده می‌باشد.

۶-۲- فاز ادغام

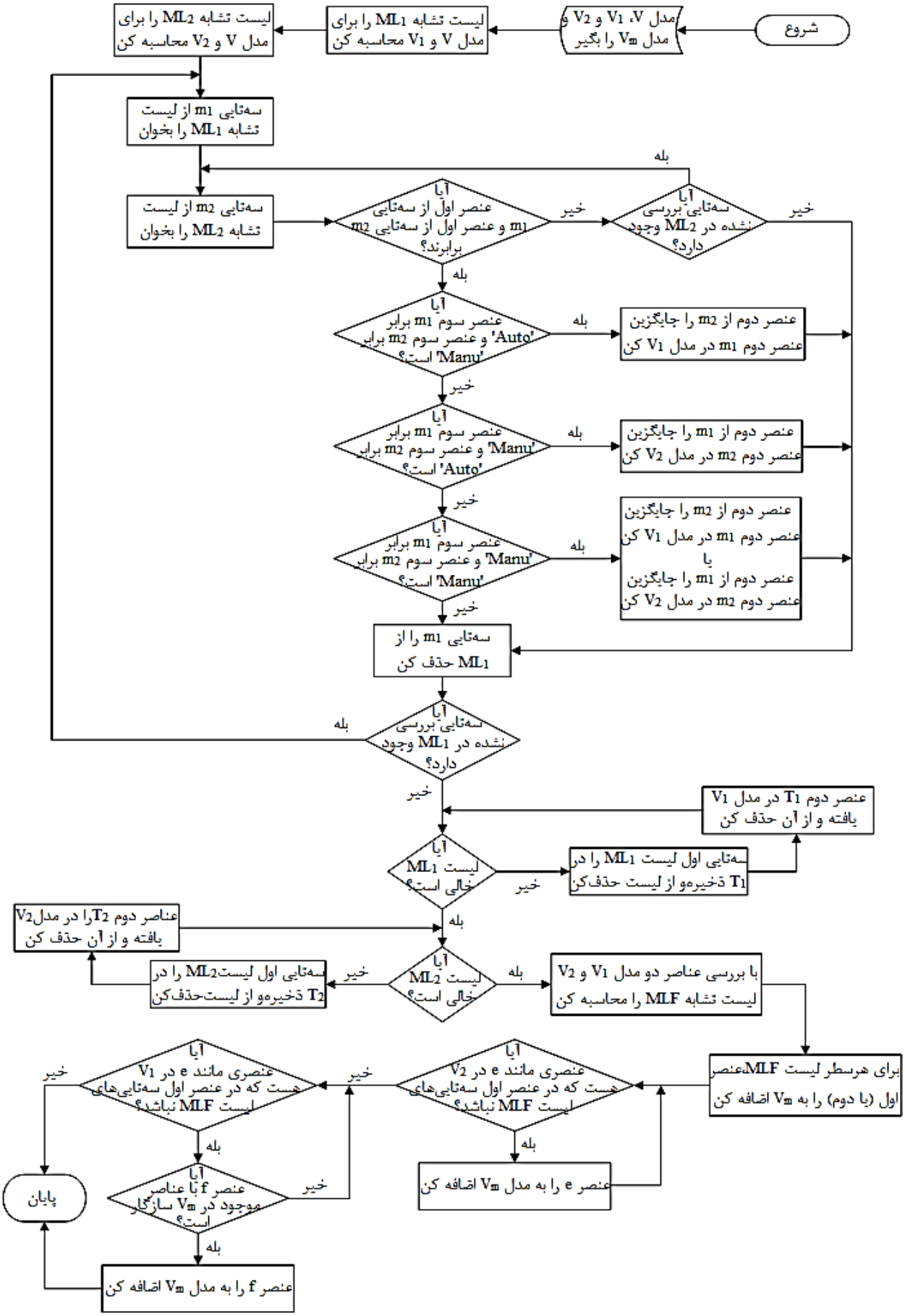
برای ایجاد نسخه‌ای یکپارچه از نسخه‌های اصلی (V)، نسخه ویرایش‌شده توسط طراح اول (V_1) و نسخه ویرایش‌شده توسط طراح دوم (V_2)، نیاز به بهره‌گیری از روشی برای ادغام می‌باشیم که با توجه به مزایای روش ادغام سه‌طرفه نسبت به دو روش دیگر، از آن برای ارائه الگوریتم پیشنهادی ادغام استفاده می‌کنیم. بر این اساس اصولی که بایستی رعایت گردد عبارت‌اند از:

- اگر عنصری از نسخه V، بدون تغییر در نسخه‌های V_1 و V_2 باشد، باید در نسخه ادغامی حاضر شود.
- اگر عنصری از نسخه V، در یکی از نسخه‌های V_1 و V_2، حذف شده باشد، نباید در نسخه ادغامی حاضر شود.
- اگر عنصری از نسخه‌ی V، در یکی از نسخه‌های V_1 و V_2، ویرایش‌شده باشد، باید عنصر ویرایش‌شده به‌جای عنصر اولیه در نسخه ادغامی حاضر شود و درصورتی‌که آن عنصر در هر دو نسخه V_1 و V_2 ویرایش‌شده باشد، اولویت با عنصر نسخه V_2 است.
- اگر عنصری به یکی از نسخه‌های V_1 و V_2، اضافه شده باشد، درصورتی‌که با سایر عناصر نسخه ادغامی سازگار باشد، باید در نسخه ادغامی حاضر شود. در غیر این‌صورت اگر نتوان ناسازگاری را برطرف نمود، بایستی از افزودن آن به نسخه ادغامی جلوگیری شود.

بر اساس اصول فوق، الگوریتم پیشنهادی در شکل ۷ نشان داده شده است که در آن با دریافت هر نسخه ویرایش‌شده و با توجه به الگوریتم مقایسه بخش ۶-۱، لیست تشابه آن نسخه و نسخه اصلی محاسبه شده و در صورت نیاز، ذهنیت طراح پیش از آغاز ادغام دریافت می‌شود. با فراهم شدن لیست تشابه اول (ML_1) برای نسخه‌های V و V_1 و لیست تشابه دوم (ML_2) برای نسخه‌های V و V_2، فاز ادغام در سه گام استنتاج، ادغام و بازسازی نسخه خروجی آغاز می‌گردد.

در گام استنتاج دو لیست تشابه دریافت‌شده از فاز مقایسه در نظر گرفته شده و به ازای هر سه‌تایی موجود در لیست ML_1، سه‌تایی‌های موجود در لیست ML_2 بررسی شده تا سه‌تایی با عنصر اول مشترک پیدا شود. در این حالت دو سه‌تایی $m_1 = (a, b_1, status)$ از لیست ML_1 و $m_2 = (a, b_2, status)$ از لیست ML_2 وجود داشته که عنصر a، عنصری از نسخه V، عنصر b_1، عنصری از نسخه V_1 و عنصر b_2، عنصری از نسخه V_2 بوده و status دارای یکی از دو مقدار 'Auto' یا 'Manu' می‌باشد. ازآنجایی‌که ترجیح بر داشتن آخرین تغییرات در نسخه خروجی ادغام می‌باشد، با توجه به مقادیر status در m_1 و m_2، اقدام به همسان‌سازی عناصر مشابه دو نسخه V_1 و V_2 نموده تا شرایط برای ادغام آماده گردد. بر این اساس سه حالت وجود داشته که نیازمند اعمال تغییر در نسخه V_1 یا V_2 می‌باشد.

- **حالت اول:** اگر عنصر سوم m_1 برابر 'Auto' و عنصر سوم m_2 برابر 'Manu' باشد، که به معنای تغییر عنصر a به عنصر b_2 در نسخه‌ی V_2 و عدم تغییر آن در نسخه V_1 است. بنابراین این عنصر در نسخه V_1 نیز با توجه به مقدار b_2 ویرایش می‌گردد.
- **حالت دوم:** اگر عنصر سوم m_1 برابر 'Manu' و عنصر سوم m_2 برابر 'Auto' باشد، که به معنای تغییر عنصر a به عنصر b_1 در نسخه V_1 و عدم تغییر آن در نسخه V_2 است. بنابراین این عنصر در نسخه V_2 نیز با توجه به مقدار b_1 ویرایش می‌گردد.
- **حالت سوم:** اگر عنصر سوم m_1 برابر 'Manu' و عنصر سوم m_2 برابر 'Manu' باشد، به معنای تغییر عنصر a به عنصر b_2 در نسخه V_2 و تغییر آن به b_1 در نسخه V_1 است. بنابراین بایستی تغییری انتخاب شود که اولویت به طراح دوم داده شده و عنصر b_1 در نسخه V_1 با توجه به مقدار b_2 ویرایش می‌گردد.



شکل ۷: الگویتم پیشنهادی برای ادغام سه‌طرفه

پس از بررسی حالت‌های مطرح‌شده و اعمال تغییرات، سه‌تایی‌های m_1 و m_2 به ترتیب از لیست ML_1 و ML_2 حذف شده و عملیات بررسی برای سایر سه‌تایی‌های موجود در لیست ML_1 ادامه یافته تا تمامی آن‌ها بررسی شوند. با پایان بررسی، سه‌تایی‌هایی که با توجه به عنصر اول معادلی نداشته‌اند، در دو لیست باقی مانده‌اند. حال برای تک‌تک این سه‌تایی‌ها از لیست ML_1، عنصر دوم را از نسخه V_1 و برای لیست ML_2، عنصر دوم را از نسخه V_2 حذف می‌نماییم. با این کار از حذف عناصری که در نسخه اصلی وجود داشته و در یکی از نسخه‌های ویرایش‌شده، حذف شده‌اند اطمینان حاصل می‌نماییم. با انجام این مرحله، گام استنتاج که نتیجه آن آماده‌سازی دو نسخه V_1 و V_2 برای گام ادغام است، به پایان می‌رسد. در گام ادغام، ابتدا بایستی دو نسخه آماده‌شده V_1 و V_2 را مقایسه نماییم. این کار بر اساس معیارهای تشابه بیان‌شده در فاز مقایسه انجام می‌شود.

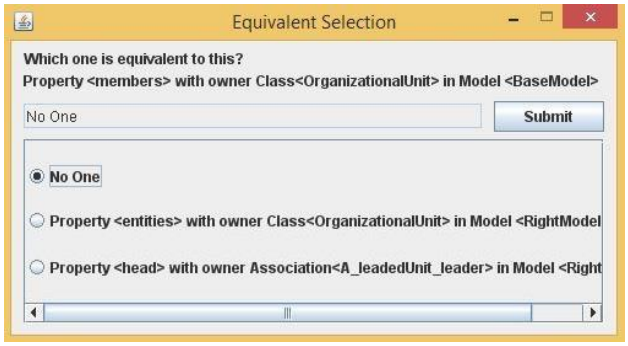
با توجه به تغییرات اعمال‌شده در نسخه‌های V_1 و V_2 در گام استنتاج، نیازی به پرسش از کاربر نبوده و به‌صورت خودکار لیست تشابه MLF، شامل سه‌تایی‌هایی به حالت $m = (b_1, b_2, \text{'Auto'})$ ایجاد می‌شود. حال به ازای هر سه‌تایی موجود در این لیست بایستی عنصر معادل b_1 و b_2 (که هر دو مشابه هستند)، به نسخه ادغامی اضافه گردد. در ادامه بایستی عناصر جدیدی که توسط طراحان به نسخه‌های V_1 و V_2 اضافه شده‌اند را به نسخه ادغامی بیفزاییم. با توجه به اولویت نسخه V_2، عناصری از این نسخه که در بین عناصر دوم از سه‌تایی‌های لیست MLF وجود نداشته و به عبارتی به نسخه ادغام اضافه نشده‌اند را به نسخه‌ی ادغامی اضافه می‌نماییم.

تا این مرحله از کار با توجه به اینکه تمامی عناصر اضافه‌شده به نسخه ادغامی، یا خود از نسخه V_2 بوده یا مشابه آن در نسخه V_2 وجود داشته است، سازگاری نسخه ادغامی حفظ شده است. ولی برای افزودن عناصر جدید نسخه V_1 به نسخه ادغامی، بایستی امکان سازگاری هر یک با عناصر موجود در نسخه ادغامی بررسی شود. به همین منظور گام بازسازی همراه با افزودن هر عنصر جدید از نسخه V_1 که در بین عناصر اول از سه‌تایی‌های لیست MLF وجود نداشته، انجام شده و در صورت سازگاری (یا امکان برقراری آن) به نسخه ادغامی اضافه می‌شود. با پایان این گام، نسخه ادغام‌شده در V_m ذخیره شده است.

۶-۳- پیاده‌سازی

یکی از ابزارهای رایگان توسعه نرم‌افزار که از کدنویسی، مدل‌سازی و توسعه تیمی پشتیبانی می‌کند، اکلیپس می‌باشد. به همین منظور به کمک بستر جاوا، افزونه‌ای تحت اکلیپس، با نام 3-Way Merger برای الگوریتم‌های بیان‌شده در بخش‌های ۶-۱ و ۶-۲ ایجاد گردید. با اجرای این افزونه، فایل‌های xmi. مربوط به نسخه‌های ورودی، شامل نسخه اصلی و نسخه‌های ویرایش‌شده، به همراه نسخه‌ای خالی برای ذخیره نتیجه ادغام دریافت شده و عملیات ادغام آغاز می‌شود.

با آغاز عملیات ادغام، ابتدا فاز مقایسه انجام شده و در صورت نیاز ذهنیت طراح دریافت می‌شود. به‌عنوان نمونه، در شکل ۵، طراح نسخه سمت راست، ارجاع members را به entities و ارجاع leader را به head تغییر داده است. شکل ۸ شیوه دریافت ذهنیت این طراح برای تعیین عنصر مشابه با ارجاع members را نمایش می‌دهد. ازآنجایی‌که entities و head هر دو، از نوع ارجاع بوده و در مقایسه خودکار عنصر مشابه‌ای برای آن‌ها یافت نشده است، به‌عنوان گزینه‌های احتمالی برای ارجاع members آورده شده‌اند.



شکل ۸: شیوه دریافت ذهنیت طراحان

پس از پایان فاز مقایسه، فاز ادغام به‌صورت خودکار اجرا شده و با ذخیره نتیجه ادغام در نسخه خروجی، پیامی مبنی بر پایان ادغام نمایش داده می‌شود.

۷- ارزیابی و مقایسه

در این بخش قصد داریم، ابتدا صحت الگوریتم پیشنهادی و ابزار تولید شده را بررسی نموده و پس از محاسبه مرتبه زمانی آن، اقدام به مقایسه الگوریتم پیشنهادی با الگوریتم‌های مشابه نماییم.

۷-۱- صحت الگوریتم

به‌منظور بررسی صحت الگوریتم مطرح‌شده، دو سناریوی آزمون در نظر گرفته شد. در سناریوی اول، فازهای مقایسه و ادغام هرکدام به‌صورت جداگانه برای نمونه‌های کوچکی که در [۴] مطرح شده است، بررسی گردید. نتایج این نمونه‌ها صحت عملکرد بخش‌های مختلف الگوریتم‌های ارائه‌شده را تأیید می‌نماید. در سناریوی دوم، کل الگوریتم به کمک محک Project_Management موردبررسی قرار گرفت که نتایج به‌دست‌آمده حاکی از پشتیبانی الگوریتم پیشنهادی از هر سه دسته تغییرات مطرح‌شده در جدول ۱ می‌باشد.

۷-۲- صحت ابزار

برای اطمینان از درستی کدهای نوشته در زبان جاوا، نزدیک به ۲۰۰ مورد آزمون برای قسمت‌های مختلف، اجرا شد که در انتها تمامی آن‌ها با موفقیت انجام شد. همچنین اجرای محک Project_Management با استفاده از ابزار ایجادشده و مقایسه نسخه ادغامی به‌دست‌آمده با نتیجه ادغام نشان‌داده‌شده در شکل ۵، حاکی از عملکرد صحیح الگوریتم پیشنهادی و ابزار ایجادشده می‌باشد.

۷-۳- محاسبه مرتبه زمانی

ازآنجایی‌که مرتبه زمانی یکی از معیارهای مهم در ارزیابی الگوریتم‌ها می‌باشد، در این بخش به بررسی اجمالی آن برای الگوریتم پیشنهادی می‌پردازیم. درصورتی‌که زمان‌های مورد نیاز برای انجام مقایسه دو عنصر، اعمال تغییر برای ویرایش یک عنصر و ادغام دو عنصر مشابه، مقداری ثابت بوده و حداکثر تعداد عناصر موجود در هر نسخه n عنصر فرض شود، مرتبه زمانی مربوط به هر فاز الگوریتم پیشنهادی عبارت است از:

- **فاز مقایسه:** در بدترین حالت به ازای هر n عنصر نسخه اول، بایستی تمامی n عنصر نسخه دوم بررسی شود درنتیجه الگوریتم مقایسه از مرتبه $O(n^2)$ خواهد بود.
- **فاز استنتاج:** با توجه به اینکه عنصری از هر نسخه تنها یک بار می‌تواند در لیست تشابه حاضر شود، بنابراین بیشترین تعداد سه‌تایی‌های موجود در دو لیست تشابه، n سه‌تایی بوده و زمان بررسی آن‌ها $O(n^2)$ می‌باشد. علاوه‌براین پس از پایان بررسی لیست‌های تشابه، درصورتی‌که هیچ سه‌تایی مشابهی یافت نشود، نیاز به انجام عملیات حذف برای هر لیست بوده که از $O(n)$ می‌باشد. بنابراین مرتبه زمانی این فاز $O(n^2)$ می‌باشد.
- **فاز ادغام:** در ابتدای این فاز بایستی دو نسخه ویرایش‌شده V_1 و V_2 مقایسه شوند که مرتبه زمانی آن $O(n^2)$ می‌باشد. پس از آن به ازای هر سه‌تایی از لیست تشابه به‌دست‌آمده بایستی یک عملیات ادغام ساده انجام گیرد، که با توجه به طول لیست که در بدترین حالت n می‌شود از مرتبه $O(n)$ خواهد بود. بنابراین، این فاز نیز از مرتبه $O(n^2)$ می‌باشد.
- **فاز بازسازی خروجی:** در بدترین حالت در این فاز، هر n عنصر نسخه V_2 در نسخه ادغامی قرار گرفته و بایستی سازگاری تمامی عناصر V_1 با تک‌تک عناصر ادغامی بررسی شود. درنتیجه مرتبه زمانی آن از $O(n^2)$ خواهد بود.

با توجه به مرتبه زمانی بیان‌شده برای فازهای مختلف، مرتبه زمانی کل الگوریتم پیشنهادی در بدترین حالت از $O(n^2)$ می‌باشد.

۷-۴- مقایسه با الگوریتم‌های مشابه

در بخش ۵، دو الگوریتم مشابه با راهکار پیشنهادی معرفی شد که به‌صورت افزونه‌ای تحت اکلیپس ارائه شده و دارای قابلیت کشف و رفع تداخل‌ها در ادغام سه‌طرفه مدل‌های یوام‌ال می‌باشند. در این قسمت به مقایسه این دو الگوریتم با راهکار پیشنهادی می‌پردازیم. بدین منظور، محک Project_Management، با استفاده از هر سه الگوریتم اجرا شده و زمان اجرای آن‌ها، با چشم‌پوشی از مدت‌زمان موردنیاز برای تعامل با کاربر، محاسبه گردید. همچنین با توجه به ویژگی‌های بیان‌شده برای این الگوریتم‌ها، معیارهای مشترکی که از آن‌ها می‌توان برای انجام مقایسه استفاده نمود، عبارت‌اند از:

- مرحله‌ای از اجرای الگوریتم که تعامل با کاربر انجام می‌شود.
- مرتبه زمانی الگوریتم
- معیار تشخیص شباهت دو عنصر
- شیوه مورد استفاده برای مقایسه
- توانایی الگوریتم در تشخیص سه نوع تغییر ممکن

جدول ۲، نتایج به‌دست‌آمده از انجام مقایسه بر اساس معیارهای مشخص‌شده را نشان می‌دهد. با توجه به دریافت مفروضات ذهنی طراحان پیش از اجرای ادغام در راهکار پیشنهادی و اجرای خودکار ادغام و عدم‌نیاز به حضور کاربر، می‌توان عدم وابستگی الگوریتم پیشنهادی به حضور کاربر در زمان اجرای ادغام را نتیجه گرفت. علاوه‌براین، با توجه به امکان دریافت نسخه‌های ورودی به فرمت xmi. که فرمتی استاندارد برای تمامی زبان‌های مدل‌سازی بوده و بایستی توسط تمامی ابزارهای مدل‌سازی پشتیبانی شود و همچنین عدم‌نیاز به ذخیره‌سازی دنباله تغییرات و استفاده از شناسه یکتا در الگوریتم پیشنهادی، می‌توان عدم وابستگی آن به ابزار مدل‌سازی را نتیجه گرفت. از طرفی، قابلیت تشخیص هر سه دسته از تغییرات مطرح‌شده در جدول ۱، نشان از جامعیت این الگوریتم در تشخیص تغییرات، علی‌رغم زمان اجرای کم‌تر آن نسبت به دو الگوریتم دیگر در شرایطی یکسان می‌باشد.

جدول۲: نتایج مقایسه الگوریتم‌های ادغام سه‌طرفه

معیار مقایسه		الگوریتم پیشنهادی	الگوریتم اول [21]	الگوریتم دوم [23]
تعامل با کاربر	پیش از اجرا	✓	×	×
	در حین اجرا	×	✓	✓
معیار شباهت	شناسه یکتا	×	×	✓
	محتوا	✓	✓	×
شیوه مقایسه	مبتنی بر حالت	✓	✓	✓
	مبتنی بر عملیات	×	×	✓
تشخیص تغییرات	افزودن عنصر	✓	✓	✓
	ویرایش عنصر	✓	✓	×
	حذف عنصر	✓	✓	✓
مرتبه زمانی در بدترین حالت		$O(n^2)$	$O(n^3)$	$O(n^2)$
زمان اجرای محک برحسب ms		۸۱۷	۱۰۱۰	۸۷۰

۸- نتیجه‌گیری

در این مقاله، الگوریتمی مبتنی بر حالت برای ادغام نسخه‌هایی از مدل یوام‌ال به روش سه‌طرفه ارائه و به‌صورت افزونه‌ای تحت اکلیپس پیاده‌سازی شد. این افزونه با تغییرات جزئی و استفاده از فرامدل مربوط به زبان مدل‌سازی موردنظر، قابل استفاده برای سایر زبان‌های مدل‌سازی نیز می‌باشد. هدف اصلی در این الگوریتم، حذف نیاز به تعامل با کاربر در حین اجرای ادغام و کسب اطلاعات موردنیاز، تنها در زمان دریافت نسخه‌های ویرایش‌شده، می‌باشد. این الگوریتم در دو فاز مقایسه و ادغام انجام می‌شود.

در فاز مقایسه که در زمان تحویل هر نسخه ویرایش‌شده اجرا می‌گردد، عناصر مشابه با نسخه اصلی محاسبه می‌شوند. این کار با

توجه به تشابه محتوای عناصر و در صورت نیاز، دریافت ذهنیت طراحی که آن تغییرات را اعمال نموده است، انجام گرفته و نیازی به وجود شناسه یکتا نمی‌باشد. در فاز ادغام که با دریافت هر دو نسخه ویرایش‌شده آغاز می‌گردد، ابتدا عملیات استنتاج برای آماده‌سازی نسخه‌ها، اجرا شده و پس از آن عملیات ادغام تک‌تک عناصر مشابه برای ایجاد نسخه ادغامی انجام می‌شود. سپس عملیات بازسازی انجام شده و علاوه‌بر افزودن عناصر غیرمشترک موردنیاز، از سازگاری نسخه ادغامی اطمینان حاصل می‌شود.

صحت افزونه تولیدی و الگوریتم پیشنهادی، به کمک اجرای محک Project_Management و آزمون‌های کوچکی که برای هر قسمت در نظر گرفته شده بود، بررسی شد. خروجی‌های به‌دست‌آمده، حاکی از عملکرد صحیح الگوریتم برای هر سه دسته از تغییرات مشترک، نامتناقض و متناقض بود. همچنین مقایسه الگوریتم پیشنهادی با الگوریتم‌های مرتبط موجود، که به‌صورت افزونه‌ای تحت اکلیپس پیاده‌سازی شده‌اند، با استفاده از معیارهای مشترکی از قبیل مرتبه‌ی زمانی، انجام شد. نتایج به‌دست‌آمده نشان از عدم وابستگی به محیط مدل‌سازی، مستقل از کاربر بودن الگوریتم پیشنهادی در زمان اجرای ادغام و جامعیت آن در تشخیص تغییرات با وجود زمان پاسخ‌دهی کم‌تر نسبت به سایر الگوریتم‌ها می‌باشد.

به‌عنوان کارهای آینده پیشنهاد می‌شود از قواعد معنایی در تشخیص تشابهات استفاده شده تا نیاز به تعامل با کاربر به‌طور کامل حذف شود. با توجه به اینکه ابزار ارائه‌شده خاص مدل‌های یوام‌ال می‌باشد ولی با تغییرات جزئی قابلیت پشتیبانی برای سایر زبان‌های مدل‌سازی را دارد، گسترش آن به‌منظور اجرا برای انواع مدل‌ها و به عبارتی ایجاد ابزار ادغام سه‌طرفه به‌گونه‌ای که مستقل از فرامدل باشد، از دیگر کارهایی است که می‌تواند در آینده انجام گیرد.

مراجع

[۱] زینب اسمعیل‌پور و اشکان سامی، «گسترش ابزارهای شناسایی الگوهای طراحی با عملگر تصحیح برچسب»، *مجله مهندسی برق تبریز*، دوره ۴۵، شماره ۲، صفحه ۱۱-۲۶، ۱۳۹۴.

[۲] زهرا اسلامی مشکنانی و اشکان سامی، «تأثیر اندازه‌های طراحی نسبت به اندازه‌های کد در بهبود کارایی سامانه‌های آزمون خودکار»، *مجله مهندسی برق تبریز*، دوره ۴۲، شماره ۱، صفحه۱۳-۲۵، ۱۳۹۱.

[3] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, Morgan & Claypool, 2012.

[4] P. Langer, *Adaptable Model Versioning based on Model Transformation by Demonstration*, PhD thesis, Vienna University of Technology, 2011.

[5] K. Altmanninger, M. Seidl, and M. Wimmer, "A survey on model versioning approaches," *Int. J. Web Inf. Syst.*, vol. 5, no. 3, pp. 271 – 304, 2009.

[6] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.*, vol. 28, no. 5, pp. 449–462, 2002.

[7] K. Altmanninger, P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "Why model versioning research is needed!? an experience report," *in Proceedings of the MoDSE-MCCM 2009 Workshop@ MoDELS*, vol. 9, pp. 1–12, 2009.

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

[9] J. Bézivin, "In Search of a Basic Principle for Model Driven Engineering," *The Eur. J. Informatics Prof.*, vol. 5, no. 2, pp. 21–24, 2004.

[10] P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel, "The Past, Present, and Future of Model Versioning," *Emerg. Technol. Evol. Maint. Softw. Model. IGI Glob*, vol. 256, pp. 410–443, 2011.

[11] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Merging Models with the Epsilon Merging Language (EML)," *in Model Driven Engineering Languages and Systems*, pp. 98 – 110, 2006.

[12] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 232–282, 1998.

[13] S. C. Barrett, *Blending State Differences and Change Operations for Metamodel Independent Merging of Software Models*, Phd thesis, Concordia University, 2011.

[14] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An Introduction to Model Versioning," *in Formal Methods for Model-Driven Engineering (SFM), LNCS 7320*, pp. 336–398, 2012.

[15] F. Schwägerl, S. Uhrig, and B. Westfechtel, "Model-based tool support for consistent three-way merging of EMF models," *in Proceedings of the workshop on ACadeMics Tooling with Eclipse - ACME '13*, pp. 1–10, 2013.

[16] M. Alanen and I. Porres, "Difference and Union of Models," *in In Proceedings of UML 2003 Conference*, pp. 2–17, 2003.

[17] K. Letkeman, "Comparing and Merging UML Models in IMB Rational Software Architect: Part 3," 2010.

[18] The Eclipse Fundation, "EMF compare," 2012. [Online]. Available: www.eclipse.org/modeling/emft/projectCompare. [Accessed: 27-Oct-2015].

[19] C. Gerth, J. M. Küster, M. Luckey, and G. Engels, "Detection and resolution of conflicting change operations in version management of process models," *Softw. Syst. Model.*, vol. 12, no. 3, pp. 517–535, 2013.

[20] S. C. Barrett, P. Chalin, and G. Butler, "Table-driven detection and resolution of operation-based merge conflicts with mirador," *in European Conference on Modelling Foundations and Applications*, pp. 329–344, 2011.

[21] B. Westfechtel, "Merging of EMF models," *Softw. Syst. Model.*, vol. 13, no. 2, pp. 757–788, 2014.

[22] A. Cicchetti, D. Di Ruscio, and A. Pierantonio, "Managing model conflicts in distributed development," *in 11th International Conference, MoDELS*, pp. 311–325, 2008.

[23] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer, "A fundamental approach to model versioning based on graph modifications: From theory to implementation," *Softw. Syst. Model.*, vol. 13, no. 1, pp. 239–272, 2014.

[24] R. Lutz, D. Wurfel, and S. Diehl, "How Humans Merge UML-Models," *in Empirical Software Engineering*, pp. 177–186, 2011.

زیرنویس‌ها

[1] Model Driven Development
[2] Artifact
[3] Version Control Systems
[4] Pessimistic Versioning
[5] Optimistic Versioning
[6] Three-Way Merging
[7] Conflict
[8] Abstract Data Type
[9] Metamodel
[10] Unified Modeling Language (UML)
[11] Raw Merging
[12] Two-Way Merging
[13] Epsilon
[14] Eclipse Modeling Framework
[15] context-free
[16] context-sensitive